Grant Agreement: 287829

Comprehensive Modelling for Advanced Systems of Systems

Compositional Analysis and Design of CML Models

Deliverable Number: D24.1

Version: 0.4

Date: March, 2012

Public Document Document

http://www.compass-research.eu

Contributors:

Marcel Oliveira, UFPE
Augusto Sampaio, UFPE
Pedro Antonino, UFPE
Rodrigo Ramos, UFPE
Ana Cavalcanti, University of York
Jim Woodcock, University of York

Editors:

Marcel Oliveira, UFPE
Augusto Sampaio, UFPE

Reviewers:

Universidade Federal de Pernambuco, University of Newcastle, University of York

Document History

Ver	Date	Author	Description
0.1	05-12-2012	Marcel Oliveira	Initial document version
0.2	05-02-2013	Marcel Oliveira	Sent for Internal Review with Contributors of Deliverable
0.3	08-03-2013	Marcel Oliveira	Sent for Internal Review with COMPASS Members
0.4	27-03-2013	Marcel Oliveira	Ready for 2nd Year Project Review

Abstract

Several compositional approaches (both for development and analysis) have been proposed as promising paradigms to deal with the ever increasing need for mastering complexity, evolution and reuse in the design of computer based systems. In order to ensure the success of a compositional reasoning method, it is essential that we trust the behaviour of the constituent parts and their combination. These can be components in component-based system development approaches, or entire systems in the more recent effort to systematise the development of Systems of Systems. Such trustworthiness is even more important in critical applications. It is crucial to verify whether Systems of Systems (SoS) satisfy some desired properties. In fact, most dysfunctional interactions are originated by classical problems in concurrent systems, such as deadlock and livelock. Unfortunately, it is at present difficult to verify important properties of industrial applications in a compositional way. Most well known industrial models, which define constituents and how they integrate, are widely based on simple, low-level granularity parts represented by syntactical interfaces, which lack behavioural information and restrict verification. Furthermore, the practice to date has been to verify and validate the system after it has been built – the system is designed, implemented and then verified and validated. The major issue is the high cost to fix a problem that is found in a late stage in development, especially when the problem requires redesigning the system to meet reliability or some other quality attribute requirement. For SoS, this is even more critical; as far as we are aware, there is no well established compositional approach for developing or reasoning about an SoS based on properties of its constituent systems.

Previously, we proposed a CSP-based correct-by-construction strategy for ensuring the preservation of properties of a system from proved properties of its interaction model and of its constituents. We consider the freedom of deadlock. Although we focus on this property, the strategy can be applied to predict other safety and liveness properties. Particularly, in Deliverable D24.4 (due in month 36) we will consider a notion of service conformance, which entails the preservation of (part of) the behaviour of the constituents systems after composition. In this document, we present the basic definitions of that model, which constitutes a generic component model that imposes the necessary constraints that characterise the components we deal with, and how they interact. As our approach applies to design a system (in terms of its components) or an SoS (from its constituent systems), we use the word components here in a broader sense, standing for components or entire system models. Each component is represented by a contract, which describes the

dynamic behaviour, interfaces and interaction points of the component. The component model also describes how components interact and how white-box can be packaged into black-box components.

We lift our previous results to provide a similar systematic approach to build trustworthy CML SoS. The main principle for lifting the approach from CSP to CML is to keep the main structure of the previous definitions and rules. The correctness of this lifting is based on two theoretical links presented in this document. We present a link between state-rich *Circus* processes and CSP processes, and a link between CML processes and *Circus* processes. Together, the two links provide a full path from CML to CSP, which enables the lifting of some theoretical results unveiled in the realm of CSP, like ours, to CML.

The reason for adopting *Circus* as an intermediate step is that a semantics for CML was available only in month 12. Also, as the semantics of both CML and *Circus* are defined in the framework of the UTP (Unifying Theories of Programming), the link between CML and *Circus* is relatively simple. Based on the results of the lifting, we explore a first example of compositional analysis of a simple ring buffer application specified in CML. The application of the approach to part of the case studies of COMPASS is planned for the deliverable D24.4 (due in month 36).

Despite being a promising approach, its practical effectiveness had not yet been quantitatively measured. In this document we explore variations of the composition rules with the notion of metadata that record information that can be used to alleviate some verification conditions during component composition. Mechanising the rule applications has required a CSP encoding of the composition rules and a process refinement characterisation of the rule side conditions. We then provide a detailed cost analysis of the approach by mechanically verifying the preservation of deadlock freedom in a stepwise construction of the dining philosophers example.

This document is an important contribution to one of the COMPASS objectives: developing compositional design and analysis techniques, based on architectural patterns (WP24), that will help to realise the potential and promise of SoS.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Motivation

Although compositional approaches (both for development and analysis) have been around for a long time [Mah90], over the last decade it has re-emerged as a promising paradigm to deal with the ever increasing need for mastering complexity, evolution and reuse in the design of computer based systems. The basic motivation for this paradigm is to replace conventional programming with the composition and configuration of reusable and independent constituents.

Nevertheless, in order to ensure the success of a compositional and reasoning method, it is essential that we trust the behaviour of the constituents and their integrations. Such trustworthiness is even more important in critical applications. For instance, avionics systems must have high reliability and continue to operate upon a failure [MJG+10], autonomous agents in a manufacturing system must correctly obey their schedule [Weh00, BGL+08]. Errors in these systems are caused not only by failures of individual constituents, but by dysfunctional interactions between non-failed constituents.

The reason of dysfunctional interactions is that real industrial constituents do not always fit together like 'Lego Pieces', or just using simple *glue code*. Integration solutions are often developed in an *ad hoc* manner, in which incompatibilities are not discovered until their side effects emerge during implementation [HGK+06]. Critical issues for system and Systems of Systems (SoS) construction are related to the design of the communication-based interaction mechanisms that permit constituents to work together [Spi04]. The correct design of these elements is critical; otherwise the system or the SoS may malfunction in subtle ways or may not work at all. This concern is even more acute when a group of constituents are put together and coordinated to accomplish a collective set of tasks [PA98]. Therefore, it is crucial to verify whether SoS satisfy some desired properties. In fact, most dysfunctional interactions are originated by classical problems in concurrent systems, such as deadlock and livelock.

Safety-related properties, including deadlock and livelock freedom, are emergent system attributes [Lev95]. In other words, these are properties that emerge from the interactions among multiple constituents, and their analysis might not reside in any system in particular. For this reason, emergent properties cannot be tested directly in an efficient compositional way.

In [Min07, MCMM08], it was shown that deciding deadlock freedom and liveness in interaction systems is NP-hard. Therefore, it is desirable to establish (stronger) conditions that are easier to test and entail the desired properties [GGMC$^+$06]. To help development, these conditions should be intrinsic to the design and implementation rules used by developers and integrators [Wal03, MH05]. In this way, an engineer, who is not an expert in analytic theory, can reason about properties of the design.

Problems are inevitable after all. It is impossible to foresee every possible situation in which a given system might be used. Problems will surely arise when two or more systems with interfaces that do not match are integrated. The sooner and more easily these problems are identified and resolved, the greater is the success of the compositional method.

Unfortunately, it is at present difficult to verify important properties of SoS in industry. Most well known industrial models, which define constituents and how they integrate, are widely based on simple, low-level granularity parts (EJB [DK06] and COM/DCOM [Mic11]). These are represented by syntactical interfaces, which lack behavioural information and restrict verifications [FG03].

Ironically, the idea of higher-level granularity models, such as Wright [All97a, ADG98], Fractal [BCL$^+$06] and SOFA [BHP06], has been still waiting for full commercial exploitation [Pla05]. Higher-level granularity models complement the syntactical information of a constituent with behaviour. The overall behaviour can be described using different styles that are usually associated to constituent, port and assembly behaviours [HJK10a]. The *protocol* represents the overall observable behaviour of the constituent. The *port-protocol* is associated to a port; it describes the behaviour of a point of interaction of the constituent. Finally, the assembly behaviour is related to the interaction between different constituents.

Nevertheless, formal description methods are getting more and more attention in the development of critical systems because of their accuracy and the use of theorem proving mechanisms [Chi09]. Much effort is devoted to the correctness of CBS [All97a, BCD02, HLL06b, Sif10, CZ07]. These works define a component model with a precise meaning, or adopt a formal notation to specify the system. This makes it possible to analyse the systems and to provide tool support in verifications. For SoS, this is even more critical; as far as we are aware, there is no well established compositional approach for developing or reasoning about an SoS based on properties of its constituent systems.

The practice to date has been to verify and validate the system or the SoS after it has been built [HLL06b, PV02, CCH+09] – the system or the SoS is designed, implemented and then verified and validated. The major issue is the high cost to fix a problem that is found in a late stage in development, especially when the problem requires redesigning the system to meet reliability or some other quality attribute requirement.

Instead of verifying the entire system or the SoS, other more promising approaches focus on iteratively identifying problems in compositions. However, in most approaches the cost of subsequent compositions is not alleviated by the results of the previous ones [ADG98, BCD02, CK96]. Every composition is taken as a monolithic system for verification, and properties of its constituting parts are not considered. Verification methods do not take advantage of the hierarchical structure of systems. In other words, these methods are not compositional, and have scalability problems by not using local analysis when this is possible.

In [RSM09], we describe a theoretical foundation for the development of correct systems, whose summary is presented in Section 3. We propose a correct-by-construction strategy for ensuring the preservation of properties of a system (in terms of its components) or an SoS (from its constituent systems) from proved properties of its interaction model and of its constituents. We consider freedom of deadlock and livelock. Although we focus on these properties, the strategy can be applied to predict other safety and liveness properties. Moreover, the ideas can be transferred to other formal models, and support the implementation of practical tools.

This approach is intended to address engineering concerns, and make the expertise on correctness available to engineers who are not experts in understanding the origin of dysfunctional interactions between non-failed constituents in the system. Moreover, we claim that a constructive approach, in opposition to *a posteriori* verification, is more suitable. It preserves quality attributes of the system by construction, and identifies problems early in the design phase. Moreover, we use local analysis, when this is possible, to scale the verifications in our approach.

To underpin this approach, in [RSM09], we also propose important design constraints. By satisfying these constraints at development, we can certainly trust the resulting system. These constraints characterise which kinds of constituents, as well as interactions, are supported in this work. To allow further verifications, we focus on behaviour-rich elements, in which not only syntactical information about operations is presented, but also their behaviour: the possible valid sequences of operations that the constituent can perform. The other constraints are the constructive constraints. They aimed to assist system evolution. We focus on notions that predict quality attributes of constituents in compositions. These notions allow checking whether the behaviours of two constituents are compatible for them to interoperate. The entire approach is based on the CSP process algebra [Ros98], which allows us to formally address property characterisation and preservation.

In [RSM10], we start by performing a study on protocols in isolation, independent of being associated to components. This study shows whether two protocol specifications are compatible to interoperate. The study considers both synchronous and asynchronous mediums, and presents test characterisations to verify such compatibility. A component model that delimits the broad outline of what constitutes a component, exposing its necessary related technical concepts and constraints, called BRIC, was also introduced in [RSM10]. As we intend that our approach apply both to design a system (in terms of its components) or an SoS (from its constituent systems), we use the word 'components' here in a broader sense, standing for components or entire system models.

1.2 Objetives

In this document, we present the basic definitions of that model, which imposes the necessary constraints that characterise the components we deal with, and how they interact. It is aligned with the concepts of other practical models [BHP06, HLL06b, MB05] and covers a wide variety of applications. Each constituent is represented by a contract, which describes its dynamic behaviour, interfaces and interaction points. The model also describes how constituent elements interact and how white-box can be packaged into black-box elements. The basic notions of this model were originally presented in [RSM09, RSM10].

Based on the definition of BRIC components, [RSM09] presents a correct-by-construction strategy for BRIC components. We presented a strategy for composition that is based on a comprehensive set of basic composition rules for BRIC components. The proposed rules can be regarded as safe steps to form a wide variety of trustworthy systems and SoS. The systematic use of these rules guarantees, by construction, the absence of deadlock and livelock. Most of the side conditions of these rules are based on the notion of port protocols. The verification using port protocols is more efficient since the (whole) behaviour of a constituent is typically much broader (if we compare the number of states and transitions) than its port protocols.

The proposed set of rules covers systems with arbitrary topologies, including those with cycles. An application of these composition rules for tree-topology architectures is presented in [RSM09]. The refinement based conformance notion, on which the rules are based on, was first presented in [RSM08].

To improve the practical application of our rules, Ramos proposed an enriched model, called BRICK [Ram11]. In this model, contracts are enriched with metadata to carry additional information useful in composition verifications. Such metadata enrich component contracts with static information (*i.e* port protocols and channels dependency) that assist the runtime environment with additional (validation) properties. Furthermore, we presented a new set of composition rules that take this metadata into consideration. The metadata of the composition is directly derived from the metadata of its constituting elements. As a result, the complexity of compositions is reduced, and the value of the method is improved.

Despite being a promising approach, its practical effectiveness was not quantitatively measured. For instance, the costs of the verification of the side conditions imposed by the composition rules was not compared with the costs of an *ad hoc* verification of the resulting composite system. In this document we explore variations of the composition rules presented in [RSM09], with the notion of metadata. Mechanising the rule applications required a novel CSP encoding of the composition rules and a process refinement characterisation of the rules side conditions. We then provide a detailed cost analysis of the approach by mechanically verifying the preservation of deadlock freedom using the dining philosophers example. The results presented here provide empirical evidence that our approach offers a gain, not only concerning a stepwise systematisation of the system construction, but also regarding the verification effort. Nevertheless, we demonstrate that this is the case only when the optimisations based on metadata are considered.

In this document, we lift the results from [RSM09] to provide a similar systematic approach to build trustworthy CML SoS. Envisioning the SoS context in which our work is inserted, our approach supports asynchronous communications. The main principle for lifting the approach from CSP to CML (via ***Circus*** [CSW03]) is to keep the main structure of the definitions and rules. Nevertheless, a thorough analysis indicated that some changes could be done to simplify the application of the approach. Furthermore, in order to reuse these results by providing a theoretical link for processes and refinement, as we explain in Section 5, we restricted our scope. Our link is limited to a subset of untimed feasible divergence free CML processes without object-oriented constructs and without undefined expressions with a limited use of predicative specifications. This theoretical link provides a mapping between *Circus* processes and CSP processes, which constitutes a very interesting piece of research that allows researchers to freely migrate results between these two formalisms. The soundness of this link amounts to a very large part of the work presented here.

The reason for adopting *Circus* as an intermediate step is that a semantics for CML was available only in month 12. Also, as the semantics of both CML and *Circus* are defined in the framework of the UTP (Unifying Theories of Programming), the link between CML and *Circus* is relatively simple. Based on the results of the lifting, we explore a first example of compositional analysis of a simple ring buffer application specified in CML. The application of the approach to part of the case studies of COMPASS is planned for the deliverable D24.4 (due in month 36).

The results presented here are an important step towards one of the COMPASS objectives: developing compositional design and analysis techniques, based on sophisticated architectural patterns (WP24), that will help to realise the potential and promise of SoS. They will foster reusability and substitutability (evolution) of components, by limiting impact and costs of changes. This also has an impact on cost of development, to ensure scalability. Here, we discuss small scale examples based on simple components. A large scale SoS example is currently under development and will be part of the final deliverable of task T2.4.1.

1.3 Overview

This document has the following structure. Sections 2 and 3 are devoted to background. The former presents the technical background of the document by providing a detailed description of the formal languages used here: CSP (2.1), *Circus* (2.2), CML (2.3) and their theoretical foundation framework, the Unifying Theories of Programming (2.4). The latter discusses the original systematic approach by presenting its basic definitions, composition rules and their extended counterparts. A novel quantitative analysis of the original approach is also included in this section.

The remaining sections constitute the main contributions of this document. In Section 4 we lift the results presented in Section 3 to provide a similar systematic approach to build trustworthy CML systems. Section 5 provides the theoretical correctness foundations of this lifting. Section 6 presents an initial evaluation of our approach on a case study originally presented in

Circus in [CSW03].

Finally, Section 7 presents our general conclusions, pointing out our main contributions. We analyse the advantages and disadvantages of our approach comparing with related works, and we discuss some topics for future work, particularly considering the scope of the Deliverable D24.4 (due in month 36).

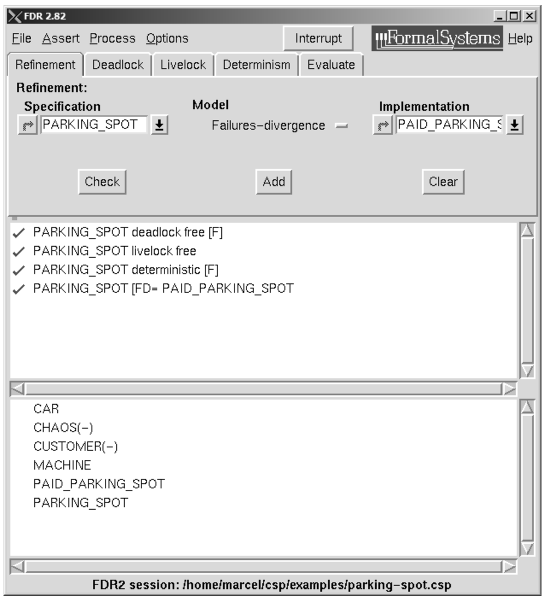
2 Technical Background

In this section, we provide the technical background of the document. In Section 2.1, we present the process algebra on which the original approach is underpinned, CSP [Hoa85, Ros98]. Next, Section 2.2 presents an extension to CSP, ***Circus*** [CSW03], which adds specification facilities in the Z [WD96] style, enabling both state and communications aspects to be captured in the same specification. This specification style is the source of inspiration to our target language, CML [WCF^{+}12], a formal specification language that integrates a state based notation (VDM++ [FL09]) and CSP, as well as Dijkstra's language of guarded commands and the refinement calculus. Finally, we present the theoretical foundations of ***Circus*** and CML, the Unifying Theories of Programming, a framework in which the theory of relations is used as a unifying basis for programming science across many different computational paradigms.

2.1 CSP

The language of CSP was first described by Hoare [Hoa85]. It is a process algebra that can be used to describe systems composed by interacting components, which are independent self-contained entities (called processes) with particular interfaces that are used to interact with the environment. In [Ros98], a new version of CSP was presented: it differs from Hoare's version only on the treatment of alphabets. It is the later version that forms the basis of FDR, a tool that model-checks a machine-processable subset of CSP, called CSP$_{\mathsf{M}}$, which is a combination of an ASCII version of CSP with an expression language inspired on functional languages. A link between the CSP and CSP$_{\mathsf{M}}$ syntaxes can be found in [Ros98]. In what follows, we briefly describe the most important CSP constructs.

The two most basic CSP processes are $STOP$ and $SKIP$; the former deadlocks, and the latter does nothing and terminates. If P is a CSP process, and a an event, then the prefixing $a \to P$ is initially able to perform only a, and after performing a it behaves as P. A boolean guard may be associated with a process: given a predicate g, if the condition g is true, the process $g \mathbin{\&} P$ behaves like P; it deadlocks otherwise. Processes $P1$ and $P2$ can be combined in sequence using the sequence operator: $P1;\ P2$. This process executes the process $P2$ after the execution of $P1$ terminates. The external choice $P1 \mathbin{\Box} P2$ initially offers events of both processes. The performance of the first event resolves the choice in favour of the process that performs


FDR 2.82
File Assert Process Options
Interrupt
FormalSystems
Help
Refinement Deadlock Livelock Determinism Evaluate
Check
Add
Clear

Refinement:
Specification: PARKING_SPOT
Model: Failures−divergence
Implementation: PAID_PARKING_S

```
✓ PARKING_SPOT deadlock free [F]
✓ PARKING_SPOT livelock free
✓ PARKING_SPOT deterministic [F]
✓ PARKING_SPOT [FD= PAID_PARKING_SPOT
```

```
CAR
CHAOS(−)
CUSTOMER(−)
MACHINE
PAID_PARKING_SPOT
PARKING_SPOT
```

FDR2 session: /home/marcel/csp/examples/parking−spot.csp

Figure 1: FDR GUI

it. Differently from the external choice, the environment has no control over the internal choice $P1 \sqcap P2$, in which the process internally (nondeterministically) resolves the choice. The sharing parallel composition $P1 \parallel[\, cs \,]\parallel P2$ synchronises $P1$ and $P2$ on the channels in the set cs; events that are not listed occur independently. Differently, in the alphabetised parallel composition $P1 \,[\![cs1 \mid cs2]\!]\, P2$, the processes $P1$ and $P2$ synchronise on the channels that are in the intersection between $cs1$ and $cs2$; events that are not in this intersection occur independently. Processes can also be composed in interleaving: in $P1 \mid\mid\mid P2$, both processes run independently. The event hiding operator $P \setminus cs$ is used to encapsulate the events that are in the channel set cs. This removes these events from the interface of P, which become no longer visible to the environment. CSP also provides finite iterated operators that can be used to generalise the binary operators of sequence, external and internal choice, parallel composition, and interleaving. A few other process constructors are available in CSP but omitted here for conciseness. Furthermore, we also omit the syntax of the expression language accepted in CSP, which can be found in [Ros98].

By way of illustration, we consider the development of a parking spot presented in Figure 2. In CSP, every channel used in the specification must be declared. For instance, *channel enter*, *leave* declares the channels *enter* and *leave*, which indicate that a customer has entered the parking spot and left it, respectively. Our abstract specification of a parking spot, *PARKING_SPOT*

$channel\ enter, leave$

$PARKING\_SPOT = enter \rightarrow leave \rightarrow PARKING\_SPOT$

$datatype\ ALPHA = a \mid b$
$datatype\ ID = Letter.ALPHA \mid unknown$
$channel\ cash, ticket, change : ID$

$MACHINE = cash?id \rightarrow ticket.id \rightarrow change.id \rightarrow MACHINE$

$CUSTOMER(id) =$
$\quad(enter \rightarrow cash!id \rightarrow$
$\quad\quad(ticket.id \rightarrow change.id \rightarrow SKIP$
$\quad\quad\Box\ change.id \rightarrow ticket.id \rightarrow SKIP));$
$\quad leave \rightarrow CUSTOMER(id)$

$PAID\_PARKING\_SPOT =$
$\quad(CUSTOMER(Letter.a)$
$\quad[\![\{\!|\ cash, ticket, change\ |\!\}]\!]$
$\quad MACHINE) \setminus \{\!|\ cash, ticket, change\ |\!\}$

Figure 2: A Simple CSP Example

only requires that two customers cannot enter in sequence; the first one must leave before the next one enters. Using FDR's assertion commands, we can verify that the abstract specification of the parking spot is deadlock free, divergence free, and deterministic. This is indicated in FDR with a ✓ on the left of the assertion in Figure 1.

Our concrete parking spot, $PAID\_PARKING\_SPOT$ is a paid version of a public parking spot with a pay and display parking permit machine that accepts cash, and issues tickets and change. First, we declare a datatype that represents simple identifications. Datatypes can be divided into two groups: basic datatypes and complex datatypes. The former is defined in terms of simple constants and the latter uses constructors that are applied to types. In Figure 2, $datatype\ ALPHA = a \mid b$ defines a datatype $ALPHA$: variables of type $ALPHA$ can assume either value a or b. On the other hand, $datatype\ ID = Letter.ALPHA \mid unknown$ defines a ID that represents identifications. The constructors $Letter$ receives an $ALPHA$ value and returns a value of type ID (for example, $Letter.a$); another possibility is the $unknown$ ID.

Next, we declare all the new channels that are used in the concrete specification. Then, we declare the *MACHINE* process, which implements the functions of issuing tickets and change, after receiving the cash. After entering the parking spot, a *CUSTOMER* must interact with the ticket *MACHINE* by inserting the *cash* into it. The *CUSTOMER* can then pick the *ticket* and the *change* in any order, and finally, *leave* the parking spot. In order to uniquely identify each customer, we parameterise the process *CUSTOMER* with an identification, which is used to identify this customer while interacting with the machine via channels *cash*, *ticket*, and *change*. This guarantees that the machine will only issue the ticket and the change to the customer who inserted the cash.

The paid parking spot is modelled by the process *PAID_PARKING_SPOT*. It is a parallel composition of the processes *CUSTOMER* and *MACHINE*; they synchronise on *cash*, *ticket* and *change*, which are hidden from the environment. The specification *PAID_PARKING_SPOT* must always allow only one customer to enter, and then to leave the parking spot. The assertion *assert PARKING_SPOT* [*FD* = *PAID_PARKING_SPOT* captures the failures divergence refinement check to be carried out.

2.1.1 CSP semantic models

CSP offers a number of semantical approaches. A process written in CSP may be understood in terms of operational semantics (where the process is transformed to a labelled transition system, with transitions representing communications); or in terms of algebraic semantics (where properties of a process – such as equivalence to some other process – may be deduced by syntactic transformations on the process text following a set of algebraic laws); or in terms of denotational semantics (where the process corresponds to a value in some mathematical model, typically a complete partial order or a complete metric space). The latter is the one of particular interest for our work.

In what follows we briefly describe the three denotational models: traces, failures and failures-divergences [Ros98].

Traces model. The traces model $\mathcal{T}$ denotes a CSP process according to its traces, which are the set of sequences of communications which the process may engage. Let $\mathcal{A}^{*\checkmark} = \Sigma^* \cup \{s \frown \langle \checkmark \rangle \mid s \in \Sigma^*\}$ be the alphabet of communications. Formally in the traces model each process is identified by a set $T \subseteq \mathcal{A}^{*\checkmark}$ that satisfies the following healthiness condition:

T1. T is nonempty and prefix-closed. This means that it always contains the empty trace $\langle\,\rangle$ and if $s \frown t \in T$ then $s \in T$.

Given a CSP process P, the traces of P are denoted as $traces(P)$. For example, $STOP$ never communicates anything: its set of traces consists only of the empty trace $traces(STOP) = \{\langle\,\rangle\}$. Furthermore, the traces of an prefix process are the traces of the prefixed process P, each prefixed with the event a first communicated and the empty trace added ($traces(a \rightarrow P) = \{\langle\,\rangle\} \cup \{\langle a\rangle \frown s \mid s \in traces(P)\}$). Details about the other constructors are presented in [Ros98].

A process C is a trace refinement of A if, and only if, it contains all traces within A.

Definition 2.1 (Traces refinement) *Let P, Q be CSP processes. P is a trace refinement of Q, written as $Q \sqsubseteq_{\mathrm{T}} P$, if and only if, $traces(P) \subseteq traces(Q)$.*

Two processes P and Q are traces-equivalent, $P \equiv_{\mathrm{T}} Q$, if $P \sqsubseteq_{\mathrm{T}} Q$ and $Q \sqsubseteq_{\mathrm{T}} P$, i.e., $traces(P) = traces(Q)$. The process $STOP$ is the most refined process in the traces model, i.e., $P \sqsubseteq_{\mathrm{T}} STOP$ for all processes P.

The traces model is the weakest of the three denotational models of CSP that we consider. In fact, the traces of internal and external choice are indistinguishable. This indicates that $traces(P)$ does not give a complete description of P, since we would like to be able to distinguish between $P \sqcap Q$ and $P \;\Box\; Q$. For example, the process $a \rightarrow SKIP$ guarantees that if the environment is prepared to engage in the event a and then terminate, then it can engage in the event a and terminate successfully. However, $a \rightarrow SKIP \sqcap a \rightarrow STOP$ does not guarantee that it can engage in the event a and terminate successfully if the environment is ready to engage in the event a and terminates. The traces model identifies both processes as they have the same traces. However, one of them guarantees that it will terminate successfully, but the other does not guarantee.

In terms of verification, the traces model can be deployed for the verification of safety conditions. That is, a process Q which is a trace refinement of a process P, will perform traces already defined in P and nothing more, i.e., $traces(Q) \subseteq traces(P)$. Safety conditions are concerned with the exclusion of traces only.

Stable failures Model. The stable failures model $\mathcal{F}$ gives a finer information about processes. For instance, it allows us to distinguish between internal and external choice (and much more). In particular, it allows us to detect deadlocked processes. A *failure* of a process is a pair (s, X), that describes a set of events X which a process can fail to accept after executing the trace s. The set X is called the *refusal* set; the process cannot perform any event in the set X no matter for how long it is offered.

The 'stable' in the model name means that the sequences represented by s are those that reach a *stable state* where no transition is chosen nondeterministically. In other words, stable states are those in which there are no choices between external and internal actions.

As an example, let us consider the following processes over the alphabet $\{a, b\}$:

$$\begin{aligned} P &= a \to STOP \;\Box\; b \to STOP \\ Q &= a \to STOP \sqcap b \to STOP \end{aligned}$$

The stable failure set of P and Q, denoted by *failures*(P) and *failures*(Q), are given by:

$$\begin{aligned} \mathit{failures}(P) = {} & \{(\langle\rangle, \{\checkmark\})\} \\ & \cup \{(\langle a\rangle, X) \mid X \subseteq \{a, b, \checkmark\}\} \\ & \cup \{(\langle b\rangle, X) \mid X \subseteq \{a, b, \checkmark\}\} \\ \mathit{failures}(Q) = {} & \{(\langle\rangle, X) \mid X \subseteq \{a, \checkmark\}\} \\ & \cup \{(\langle\rangle, X) \mid X \subseteq \{b, \checkmark\}\} \\ & \cup \{(\langle a\rangle, X) \mid X \subseteq \{a, b, \checkmark\}\} \\ & \cup \{(\langle b\rangle, X) \mid X \subseteq \{a, b, \checkmark\}\} \end{aligned}$$

Here, P and Q have different failures, i.e., the stable failures model $\mathcal{F}$ can distinguish between internal and external choice. The failures of P records that initially (after the trace $s = \langle\rangle$) the process cannot refuse either a or b. The process Q has two initial invisible actions τ to choose from. After performing them, it reaches stable states, where it can perform either a or b separately, and refuse b or a respectively. The failure of Q does not record any information about the initial state, but only information about the stable states.

Observe that it is by no means inevitable that every trace of a process has failure: it may never stop performing τ actions. So, as not all traces of a process are present in its failures, a process in the $\mathcal{F}$ model is represented not only by its stable failures, but also by its traces. Formally, in the stable failures model, each process P is modelled by a pair (T, F), denoting $T = \mathit{traces}(P)$ and $F = \mathit{failures}(P)$, where $T \subseteq \Sigma^{*\checkmark}$ and $F \subseteq \Sigma^{*\checkmark} \times \mathbb{P}(\Sigma^{*\checkmark})$, satisfying the following healthiness conditions (where s, t range over Σ^* and X, Y over $\mathbb{P}(\Sigma^{\checkmark})$):

T1. T is non-empty and prefix closed.

T2. $(s, X) \in F \Rightarrow s \in T$. This asserts that all traces performed by the failures should be recorded in the traces component T. In other words it establishes consistency between the traces component and the failures component.

T3. $s \frown \langle \checkmark \rangle \in T \Rightarrow (s \frown \langle \checkmark \rangle, X) \in F$. If a trace terminates successfully by producing $\checkmark$, then it should refuse all events in $\Sigma^{\checkmark}$ at the stable state after $s \frown \langle \checkmark \rangle$.

F2. $(s, X) \in F \wedge Y \subseteq X \Rightarrow (s, Y) \in F$. This asserts that in a stable state if a set X is refused, then any subset Y of X should also be refused.

F3. $(s, X) \in F \wedge \forall a : Y \bullet s \frown \langle a \rangle \notin T \Rightarrow (s, X \cup Y) \in F$. This asserts that if a process P can refuse the set X of events in some stable state, then the same state must also refuse any set of events Y that the process can never reach.

F4. $s \frown \langle \checkmark \rangle \in T \Rightarrow (s, \Sigma) \in F$. This asserts that if we have any terminating trace $s \frown \langle \checkmark \rangle$, these should refuse Σ at the stable state after s.

For example, $STOP$ initially refuses to communicate anything.

$$\mathit{failures}(STOP) = \{(\langle \rangle, X) \mid X \subseteq \Sigma^{\checkmark}\}$$

Furthermore, initially the prefix process cannot refuse the prefixing event. Details about the other constructors are presented in [Ros98].

$$\begin{aligned}\mathit{failures}(a \to P) = {} & \{(\langle \rangle, X) \mid a \notin X\} \\ & \cup \{(\langle a \rangle \frown s, X) \mid (s, X) \in \mathit{failures}(P)\}\end{aligned}$$

A process C is a stable failures refinement of A if, and only if, it contains all traces within A and presents less stable failures; it refuses less communications.

Definition 2.2 (Stable failure refinement) *Let P, Q be CSP processes. P is a stable failure refinement of Q, written as $Q \sqsubseteq_F P$, if, and only if: $\mathit{traces}(P) \subseteq \mathit{traces}(Q) \wedge \mathit{failures}(P) \subseteq \mathit{failures}(Q)$.*

In other words, if every trace s of Q is possible for P and every refusal after this trace is possible for P, then Q can neither accept an event nor refuse unless P does. Two processes P and Q are stable failure-equivalent, $P \equiv_{\mathrm{F}} Q$, if $\sqsubseteq_{\mathrm{F}} Q$ and $Q \sqsubseteq_{\mathrm{F}} Q$, i.e., $\mathit{traces}(P) = \mathit{traces}(Q)$ and $\mathit{failures}(P) = \mathit{failures}(Q)$. The bottom element in $\sqsubseteq_{\mathrm{F}}$ is $\Sigma^{*\checkmark}, \Sigma^{*\checkmark} \times \mathbb{P}(\Sigma^{*\checkmark}))$, while its top element is $(\langle\,\rangle, \emptyset)$.

An important phenomenon captured by $\mathcal{F}$ is deadlock. Deadlock is a phenomenon pertaining to networks of communicating processes which occur when two processes cannot agree to communicate with each other, thus the whole system becomes permanently frozen. This is potentially catastrophic in safety-critical computing applications. A network that can never exhibit deadlock is said to be deadlock-free.

In CSP deadlock is represented by the process $STOP$, which can perform only the empty trace, and after the empty trace the process $STOP$ refuses to engage in any event. In CSP, a process P is considered to be *deadlockfree*, if the process P after performing a trace s never becomes equivalent to the process $STOP$.

Definition 2.3 (Deadlock-free process) *A process P is deadlock-free in CSP if, and only if,* $\forall\, s : \Sigma^* \bullet (s, \Sigma^{\checkmark}) \notin \mathit{failures}(P)$

This definition is justified, as in the model $\mathcal{F}$ the set of stable failures is required to be closed under the subset-relation (**F2**). In other words: Before termination, the process P can never refuse all events; there is always some event that P can perform. Moreover, the stable failure refinement notion preserves the deadlock-freedom of a process. That is, if P is deadlock free and $P \sqsubseteq_{\mathrm{F}} Q$, then Q is deadlock free.

From the definition of deadlock-free, an interesting lemma about deadlock-freedom in parallel synchronisations is described below.

Lemma 2.1 *Let P and Q be divergence-free CSP processes. Then $P \parallel Q$ deadlocks if, and only if:*

$$\exists (t, X) : \mathit{failures}(P) \bullet (t, \Sigma \setminus X) \in \mathit{failures}(Q)$$

From the lemma above, it is possible to formulate an important observation about how process should communicate in order to preserve deadlock-freedom: one process can never refuses all events that the other can perform. For instance, consider that X is a maximum refusal of P, then P can perform events within $\Sigma^{\checkmark} \setminus X$. From the lemma above, in order to avoid deadlock, Q cannot refuse such events.

Failures/divergences Model. The failures/divergence model gives us the most satisfactory representation for analysing liveness and safety properties of a CSP process; it allows us to detect not only deadlocked, but also live-locked processes. Furthermore, it has long been taken as the 'standard' model for CSP.

A process diverges, if it reaches a state from which it may forever compute internally through an infinite sequence of invisible actions. This is clearly a highly undesirable feature of the process, described by as 'even worse than deadlock' [Hoa85]. Livelock may invalidate certain analysis methodologies, and is often caused by a bug in the modelling. However the possibility of writing down a divergent process arises from the presence of two crucial constructs: *hiding* and *ill-formed* recursive processes. For instance, consider the processes $P = P$ and $Q = (a \rightarrow Q) \setminus \{a\}$. Q converts the external event a into an internal action τ. Therefore, Q indefinitely performs internal actions, which leads to a divergence. As a consequence, Q and P have the same behaviour in the failures-divergences model. The CSP process **div** (the same of Q, in our example) represents the livelock phenomenon: immediately, it can refuse every event, and it diverges after any trace.

In the failures/divergence model, the processes are represented by two sets of behaviours: the failures and the divergences. The divergences of a process are the finite traces on which the process can perform an infinite sequence of internal (invisible) actions. So, each process P is modelled by the pair $(\mathit{failures}_\perp(P), \mathit{divergences}(P))$, where:

- $\mathit{divergences}(P)$ is the (extension-closed) set of traces s on which a process can diverge. Thus, $\mathit{divergences}(P)$ contains not only the traces s on which P can diverge, but also all extensions $s \frown t$ of such traces;
- $\mathit{failures}_\perp(P) = \mathit{failures}(P) \cup \{(s, X) \mid s \in \mathit{divergences}(P)\}$.

Formally the failures/divergences model $\mathcal{FD}$ is defined to be the pairs $(F_\perp, D)$ satisfying the following healthiness condition, where s, t range over $\Sigma^{*\checkmark}$, and X, Y range over $\mathbb{P}(\Sigma^{\checkmark})$:

F.1. $\mathit{traces}_\perp(P) = \mathit{traces}(P) \cup \mathit{divergences}(P)$ is non-empty and prefix closed.

F.2. $(s, X) \in F \wedge Y \subseteq X \Rightarrow (s, Y) \in F$.

F.3. $(s, X) \in F \wedge (\forall a \in Y \bullet s \frown \langle a \rangle \notin \mathit{traces}_\perp(P)) \Rightarrow (s, X \cup Y) \in F$.

F.4. $s \frown \langle \checkmark \rangle \in \mathit{traces}_\perp(P) \Rightarrow (s, \Sigma) \in F$.

D.1. $s \in D \cap \Sigma^* \wedge t \in \Sigma^{*\checkmark} \Rightarrow s \frown t \in D$.

D.2. $s \in D \Rightarrow (s, X) \in F$. This adds all divergences-related failures of F.

D.3. $s^\frown\langle\checkmark\rangle \in D \Rightarrow s \in D$. This ensures that we do not distinguish between how processes behave after successful termination.

A process C is a failures/divergence refinement of A if, and only if, it contains all failures and divergences of A: it refuses less communications and diverges in less occasions.

Definition 2.4 (Failures/divergences refinement) *Let P, Q be CSP processes. P is a failures-divergences refinement of Q, written as $Q \sqsubseteq_{\mathrm{FD}} P$, if and only if, $failures_\perp(P) \subseteq failures_\perp(Q) \wedge divergences(P) \subseteq divergences(Q)$.*

Two processes P and Q are failures-divergences equivalent, $P \equiv_{\mathrm{FD}} Q$, if $P \sqsubseteq_{\mathrm{FD}} Q$ and $Q \sqsubseteq_{\mathrm{FD}} P$, i.e., $failures_\perp(P) = failures_\perp(Q)$ and $divergences(P) = divergences(Q)$. The process **div** is the least refined process in the failures/divergence model. Then, a process is said to be free of divergence (or livelock free) if after carrying out a sequence of events, its denotation is different from **div**.

It is consensual that the failures-divergences model gives us the most satisfactory representation for analysing liveness and safety properties of a CSP process. However, when we look into the mathematical theory of how divergences are calculated, it turns out that seeing accurately what a process can do after it has already been able to diverge is very difficult, and not really worth the effort [Ros98]. By combining traces with stable failures (which is in fact the failures part of the failures-divergences model), it is possible to see beyond any divergence by ignoring divergences altogether. Moreover, it is sometimes advantageous to analyse a divergence-free process P by placing it in a context in which it may diverge as the result of hiding some set of actions; this only works when the traces and stable failures in this context are not influenced by these divergences.

For instance, the process $P = (a \rightarrow P \;\Box\; b \rightarrow P) \setminus \{\!|\; b \;|\!\}$ diverges in its initial state. The hiding operation converts the external choice into an internal choice. Therefore, the process internally chooses between the external event a and an internal action resulted from hiding b. As a consequence, P may indefinitely perform internal actions, which in the failures-divergences model leads to divergence.

As we will see in Section 3, in our formalisation of some notions, it is not convenient that certain hidden events result in divergence. For example, our intention is that the communication protocols of divergence-free components are also divergence-free processes, even after hiding all events not in the protocol interface.

Therefore, we assume in this work that basic components are divergence-free and deadlock-free, and use the semantic models presented here in verifications to ensure that such problems are not introduced in the system formed by these components. The failures model is used in local analysis, in which the involved processes are divergent-free and the applied operators are known for not introducing such a problem. The failures/divergence model is used in verifications about the compositionality of strategy proposed here, checking theirs traces, failures and divergences.

2.2 *Circus*

Circus [CSW03] is a language that is suitable for the specification of concurrent and reactive systems; it also has a theory of refinement associated to it. Its objective is to give a sound basis for the development of concurrent and distributed system in a calculational style like that of [Mor94].

Circus is based on imperative CSP [Ros98], and adds specification facilities in the Z [WD96] style. This enables both state and communications aspects to be captured in the same specification, as in [SD01]. In the same way as Z specifications, ***Circus*** programs are formed by a sequence of paragraphs. Each of these paragraphs can either be a Z paragraph [Spi92], a definition of channels, a channel set definition, or a process declaration.

We illustrate the main constructs of ***Circus*** using the specification of a simple register (Figure 3). It is initialised with zero, and can store or add a given value to its current value. It can also output or reset its current value. The specification is composed of seven paragraphs.

All the channels that are used within a process must be declared. In a channel declaration, we declare the name of the channel and the type of the values it can communicate. However, if the channel does not communicate any value, but it is used only as a synchronising event, its declaration contains only its name; no type is defined. A channel declaration may declare more than one channel of the same type. In this case, instead of a single channel name, we have a comma-separated list of channel names. This is illustrated in Figure 3 by the declaration of channels *store*, *add*, and *out*.

Generic channel declarations introduce a family of channels. For instance, the declaration **channel** $[T]\ c : T$ declares a family of channels c. For every actual type S, we have a channel $c[S]$ that communicates values of

$$
\begin{array}{l}
\mathbf{channel}\ store, add, out : \mathbb{Z} \\
\mathbf{channel}\ result, reset \\
\mathbf{process}\ Register \mathrel{\widehat=} \\
\quad \mathbf{begin\ state}\ RegSt \mathrel{\widehat=} [value : \mathbb{Z}] \\
\quad RegCycle \mathrel{\widehat=}\ \begin{array}[t]{l} store?newValue \rightarrow value := newValue \\ \Box\ add?newValue \rightarrow value := value + newValue \\ \Box\ result \rightarrow out!value \rightarrow Skip \\ \Box\ reset \rightarrow value := 0 \end{array} \\
\quad \bullet\ value := 0;\ (\mu\, X \bullet RegCycle;\ X) \\
\quad \mathbf{end} \\[1ex]
\mathbf{channel}\ read, write : \mathbb{Z} \\
\mathbf{process}\ SumClient \mathrel{\widehat=} \\
\quad \mathbf{begin} \\
\quad ReadValue \mathrel{\widehat=} read?n \rightarrow reset \rightarrow Sum(n) \\
\quad Sum \mathrel{\widehat=}\ \ n : \mathbb{Z} \bullet \begin{array}[t]{l} (n = 0)\ \&\ result \rightarrow out?r \rightarrow write!r \rightarrow Skip \\ \Box\ (n \neq 0)\ \&\ add!n \rightarrow Sum(n-1) \end{array} \\
\quad \bullet\ \mu\, X \bullet ReadValue;\ X \\
\quad \mathbf{end} \\
\mathbf{chanset}\ RegAlphabet == \{\!|\ store, add, out, result, reset\ |\!\} \\
\mathbf{process}\ Summation \mathrel{\widehat=} \\
\quad (SumClient\ [\![\ RegAlphabet\ ]\!]\ Register) \setminus RegAlphabet
\end{array}
$$

Figure 3: A Simple Register

type S. Channels can also be declared using schemas that group channel declarations, but do not have a predicate part. This follows from the fact that the only restriction that may be imposed on a channel is the type it communicates.

We may introduce sets of previously defined channels in a **chanset** paragraph. In this case, we declare the name of the set and a channel-set expression, which determines the channels that are members of this set. In our example, we declare the alphabet of the *Register* as the channel set *RegAlphabet*. These are the channels that can be used to interact with this process.

The declaration of a process is composed of its name and its definition. Furthermore, like channels, processes may also be declared generic. In this case, the declaration introduces a family of processes.

A process is specified as a (possibly) parametrised process, or as an indexed process. If a process is parametrised or indexed, we first have the declaration of its parameters. Afterwards, following a $\bullet$, in the case of parametrised processes, or a $\odot$, in the case of indexed processes, we have the declaration of the process body. In both cases, the parameters may be used as local variables in the definition of the process. If the process is not parametrised, we have only the definition of its body.

A process may be explicitly defined, or it may be defined in terms of other processes (compound processes). An explicit process definition is delimited by the keywords **begin** and **end**; it is formed by a sequence of process paragraphs and a distinguished nameless main action, which defines the process behaviour, in the end. Furthermore, in ***Circus*** we use the Z notation to define the state of a process. It is described as a schema paragraph, after the keyword **state**. Process *Register* in Figure 3 is defined in this way. The schema *RegState* describes the internal state of the process *Register*: it contains an integer *value* that stores its value. The behaviour of *Register* is described by the unnamed action after a $\bullet$. The process *Register* has a recursive behaviour: after its initialisation, it behaves like *RegCycle*, and then recurses.

Processes may also be defined in terms of other previously defined processes using the process name, CSP operators, iterated CSP operators, or indexed operators, which are particular to ***Circus*** specifications.

Processes P_1 and P_2 can be combined in sequence using the sequence operator: $P_1;P_2$. This process executes the process P_2 after the execution of P_1 terminates. The external choice $P_1 \Box P_2$ initially offers events of both processes. The performance of the first event resolves the choice in favour of the process that performs it. Differently from the external choice, the environment has no control over the internal choice $P_1 \sqcap P_2$, in which the process internally (nondeterministically) resolves the choice.

The parallel operator follows the alphabetised parallel operator approach adopted by [Ros98]; we must declare a synchronisation channel set. For instance, in $P_1 \lbrack\!\lbrack cs \rbrack\!\rbrack P_2$ the processes P_1 and P_2 synchronise on the channels in the set cs; events that are not listed occur independently. By way of illustration, the process *Summation* in Figure 3 reads a value n through channel *read*, interacts with a *Register*, and outputs the value of $\sum_{i=1}^{n} i$ through channel *write*. It is declared as a parallel composition of processes *Register* and its client *SumClient*; they synchronise on the set of events *RegAlphabet*.

Processes can also be composed in interleaving. For instance, a process

$RegisterTwice$ that represents two $Register$s running independently can be defined as the composition $Register \interleave Register$. However, an event $reset$ leads to a non-deterministic choice of which $Register$ process of the interleaving actually starts: one of the processes resets, and the other one does not.

The event hiding operator $P \setminus cs$ is used to encapsulate the events that are in the channel set cs. This removes these events from the interface of P, which become no longer visible to the environment. For instance, the process $Summation$ encapsulates the interaction between the processes $Register$ and $SumClient$ ($RegAlphabet$); the only ways to interact with $Summation$ are via the channels $write$ and $read$.

As with CSP, ***Circus*** provides finite iterated operators that can be used to generalise the binary operators of sequence, external and internal choice, parallel composition, and interleaving. Furthermore, we may instantiate a parametrised process by providing values for each of its parameters. For instance, we may have either $P(v)$, where $P \mathrel{\widehat=} (x : T \bullet Proc)$, or, for reasoning purposes, we can write directly $(x : T \bullet Proc)(v)$. Apart from sequence, all the iterated operators are commutative and associative. For this reason, there is no concern about the order of the elements in the type of the indexing variable. However, for the sequence operator, we require this type to be a finite sequence. As expected, the process $\mathbin{;}\ x : T \bullet P(x)$ is the sequential composition of processes $P(v)$, with v taken from T in the order that they appear.

Circus introduces a new operator that can be used to define processes. The indexed process $i : T \odot P$ behaves exactly like P, but for each channel c of P, we have a freshly named channel c_i. These channels are implicitly declared by the indexed operator, and communicate pairs of values: the first element, the index, is a value i of type T, and the second element is the value of the original type of the channel. An indexed process P can be instantiated using the instantiation operator $P\lfloor e \rfloor$; it behaves just like P, however, the value of the expression e is used as the first element of the pairs communicated through all the channels.

For instance, we may define a process similar to the previously defined $RegisterTwice$, in order to have the same process that represents two $Register$s running independently, but with an identification of which process is reset. In order to interact with the indexed process $IndexRegister \mathrel{\widehat=} i : \{1, 2\} \odot Register$, we must use the channels $store\_i$, add_i, $result\_i$, out_i and $reset\_i$. We may instantiate the process $IndexRegister$: the process $IndexRegister\lfloor 1 \rfloor$, for instance, outputs pairs through channel out_i whose first elements are 1 and the second elements are the values stored in the register. It may be restarted by sending the value 1 through the channel $reset\_i$. Similarly, we have the process $IndexRegister \lfloor 2 \rfloor$. Finally, we have the process presented below that represents a pair of registers: the first element of the pairs identifies the register.

$$RegisterTwiceId \mathrel{\widehat=} IndexRegister \lfloor 1 \rfloor \interleave IndexRegister \lfloor 2 \rfloor$$

The renaming operator $P[oldc := newc]$ replaces all the communications that are done through channels $oldc$ by communications through channels $newc$, which are implicitly declared, if needed. Usually, indexing and renaming are used in conjunction, as in the redefinition of the process $RegisterTwice$ presented below.

$$\begin{array}{l} RegisterTwice \mathrel{\widehat=} \\ \qquad RegisterTwiceId \left[\begin{array}{lcl} store\_i, add\_i, & & storeid, addid, \\ result\_i, out\_i, & := & resultid, outid, \\ reset\_i & & resetid \end{array} \right] \end{array}$$

We may also combine instantiations of an indexed process using the iterated operators. For example, we may redefine the process $RegisterTwiceId$ as $\interleave\, i : \{1, 2\} \bullet Register \lfloor i \rfloor$. The same characteristics and restrictions still apply to the iterated operators.

Finally, generic processes may be instantiated: the expression $P[T]$ instantiates a generic process named P using the type T.

When a process is explicitly defined, besides the definitions of the state and the main action, we have in its body Z paragraphs, definitions of (parametrised) actions, and variable sets definitions; they are used to specify the main action of the process.

As with processes, an action may be parametrised, in which case we have the declaration of the parameters followed by a $\bullet$, and then, the body of the action. An action can be a schema expression, a guarded command, an invocation to a previous defined action, or a combination of these constructs using CSP operators. Furthermore, state components and local variables may be renamed; however, no channel name can be changed.

Three primitive actions are available in ***Circus***: $Skip$, $Stop$, and $Chaos$. The action $Skip$ does not communicate any value or changes the state: it terminates immediately. The action $Stop$ deadlocks, and the action $Chaos$ diverges.

The prefix operator is standard. However, a guard construction may be associated with it. For instance, given a Z predicate p, if the condition p is *true*, the action $p \mathbin{\&} c?x \rightarrow A$ inputs a value through channel c and assigns it to the variable x, and then behaves like A, which has the variable x in scope. If, however, the condition p is *false*, the same action deadlocks. Such enabling conditions like p may be associated with any action. Predicates may also be associated with an input prefix. For instance, a communication $c?x : p$ will only happen when a value of the type of the channel c that satisfies the predicate p is communicated.

The action *Sum* in the process *SumClient* (Figure 3) exemplifies the output prefix operator. While the number n is different from 0, this action requests the *Register* to add a value to its current value by outputting n through channel *add*. Finally, when n reaches 0, it requests the *result* from the *Register*, reads it from channel *out*, and writes it to channel *write*.

All the free variables of an action must be in scope in the containing process. All actions are in the scope of the state components. Input communications introduce new variables into scope, which may not be used as targets of assignments.

The CSP operators of sequence, external and internal choice, parallel, interleaving, and hiding may also be used to compose actions. However, differently from processes, at the level actions, recursive definitions (μ) are also available.

Our *Register*, as previously described, has a recursive behaviour. Its cycle, the action *RegCycle*, is an external choice: values may be stored or accumulated, using channels *store* and *add*; the result may be requested using channel *result*, and output through *out*; finally, the register may be reset through channel *reset*.

At the level of actions, the parallel and the interleaving operators are slightly different from that of CSP in [Ros98] and [Hoa85]. In order to avoid conflicts in the access to the variables in scope, parallel composition and interleaving of actions must also declare two disjoint sets (that may partition) of variables in scope: state components, and input and local variables. In $A_1 \llbracket ns_1 \mid cs \mid ns_2 \rrbracket A_2$, both A_1 and A_2 have access to the initial values of all variables in ns_1 and ns_2, but A_1 may modify only the values of the variables in ns_1, and A_2, the values of the variables in ns_2. Besides, the actions A_1 and A_2 synchronise on the channels in the set cs.

Parametrised actions can be instantiated: for instance, we can have the action $A(x)$, if A is a previously defined single-parametrised action; we can also have an instantiation of the form $(x : T \bullet A)(x)$.

As for processes, the iterated operators for sequence, external and internal choice, parallel, and interleaving can also be used in order to generalise the corresponding operators.

Actions may also be defined using Dijkstra's guarded commands [Dij76]. An action can be a (multiple) assignment, or a guarded alternation. For instance, we store a value in the *Register* using the assignment $value := newValue$. Variable blocks can also be used in an action specification. In the interest of supporting a calculational approach to development, an action can also be written as a specification statement in the style of Morgan's refinement calculus [Mor94]. We adopt the syntactic sugaring $\{pre\}$ for specification statements : $[pre, true]$ (assumptions). In the same way, the coercion $[post]$ is a syntactic sugaring for : $[true, post]$. The invocation of substitutions by value, result, or by value-result, as those presented in [Cav97], are also available in ***Circus***.

Circus and CML, which is the subject of the next section, are indeed very similar languages. Both languages are based on a language for data-modelling and CSP. The former uses Z as its data language and the latter uses the Vienna Development Method (VDM) [Jon90]. In addition, CML also includes constructs for object orientation based on VDM++ [FL09] and an object-oriented extension of ***Circus*** [CSW05b], and constructs for time modelling based on Timed CSP and a timed extension of ***Circus*** [SCHS10].

2.3 CML

The COMPASS modelling language (CML) [WCF$^+$12] is a formal specification language that integrates a state based notation (VDM++) and a process algebraic notation (CSP [Hoa85]), as well as Dijkstra's language of guarded commands and the refinement calculus. It supports the specification and analysis of state-rich distributed specifications. Additionally, CML supports step-wise development by means of algebraic refinement laws. The soundness of the refinement laws is established with respect to the formal semantics of CML, defined in Unifying Theories of Programming [HJ98]. CML is still under development, with a COMPASS tool and several analysis plug-ins currently in production [CML$^+$12]. In particular, tool support for CML will include a parser, a type-checker, a simulator, a theorem prover, a model-checker and a refinement editor.

In the remainder of this chapter, we introduce CML by means of a specification of a simple clock, and provide extensions to the clock example to illustrate features of the CML language. For more details on CML, refer to [WCF+12, WCC+12].

Initially, we specify a simple clock whose only observable behaviour is a synchronisation on a channel `tick`.

```
channels tick
```

Internally, the clock has a state variable `s` that records the number of seconds (marked by `tick`) that have elapsed, and has two operations defined: `Init()` and `increment`. The first simply initialises the state with 0, and the second adds one to the state component. The state is captured by the following class declaration.

```
class ClockSt =
begin
    state
        public s: nat
    initial
        public Init()
        frame wr s
        post s = 0
    operations
        public increment()
        frame wr s
        post s = s~ + 1
end
```

The `frame` keyword in the declaration of operations specifies the state components that can be read and written by the operation. In the case of the `Init` operation, the state component `s` can be written by `Init`. The `post` keyword specifies the post-condition of the operation. In the case of `Init`, the post-condition states that the state component `s` (after the operation) is equal to zero. The post-condition of the operation `increment` equates the state component `s`, after the operation, to the sum of its initial value (`s~`) and one.

Our simple clock initialises its state, waits for one time unit, which we take to mean one second, increments its counter and synchronises on `tick`. This is specified by the following process declaration.

```
process SimpleClock =
begin
    state
        c: ClockSt
    actions
        Ticking = Wait 1; c.increment(); tick -> Skip
    @ c.Init(); mu X @  Ticking; X
end
```

The simple clock is a process that declares a state and a number of actions. The state, in this case, is formed by a single state component `c` of type `ClockSt`. The actions include `Ticking` and the action started by `@`. The latter is a mandatory main action that defines the behaviour of the process; in this case, it simply initialises the state by calling the operation `Init()` of the state component `c` and recursively (`mu`) calls the action `Ticking`. This action waits for one time unit, increments the internal counter and synchronises on the channel `tick`.

Our initial specification of the clock is extremely simple, the only observable event is the synchronisation on `tick`. It might be interesting to have a clock that takes advantage of its internal counter and supplies information about how many seconds, minutes, hours and days have elapsed.

We now extend our simple clock to include this additional functionality. First, we declare four additional channels that communicate a natural number. They are used to query the seconds, minutes, hours and days that have elapsed.

```
channels second, minute, hour, day: nat
```

The new clock specification is similar to the simple clock; it declares the state of the process as the component `c` of type `ClockSt`, but additionally defines three functions: `get_minute`, `get_hour` and `get_day`. They take the number of seconds recorded in the state, and calculate, respectively, the equivalent number of minutes, hours and days.

```
process Clock =
begin
    state c: ClockSt
    functions
        get_minute(s: nat) m: nat
        post m = s/60

        get_hour(s: nat) h: nat
        post h = get_minute(s)/60

        get_day(s: nat) d: nat
        post d = get_hour(s)/24
```

The ticking action remains the same as before, but we add a new action, `Interface`, that provides the extra functionality.

```
    actions
        Ticking = Wait 1; c.increment(); tick -> Skip
        Interface = second!(c.s) -> Interface
                 [] minute!(get_minute(c.s)) -> Interface
                 [] hour!(get_hour(c.s)) -> Interface
                 [] day!(get_day(c.s)) -> Interface
```

This action simply offers a choice (`[]`) of communication over the channels `second`, `minute`, `hour` and `day`, and recurses. Each communication outputs (outputs are indicated by `!` after a channel name) the appropriate value calculated using the functions previously defined.

Now, the main action of the new clock is slightly different. It first initialises the state as usual, but instead of offering `Ticking` alone, it composes `Ticking` in parallel with the recursive action `Interface` with the option of interrupting (/\) `Interface` with a synchronisation on `tick`. The parallel operator `[| ns1 | cs | ns2|]` contains a set of events `cs` on which the two parallel actions synchronise, and two name sets `ns1` and `ns2` that partition the state of the process and indicate which state components can be updated by the left (`ns1) and right (\verbns2!`) parallel actions. In our example, the action `Ticking` can update the state component `c` and the right parallel action does not update the state. The parallel actions synchronise on the channel `tick`.

The two parallel action synchronise on the channel `tick`.

```
    @ c.Init(); mu X @ (
                Ticking
                    [| {c} | {|tick|} | {} |]
                (Interface/\tick -> Skip)
              ); X
end
```

While `Ticking` is waiting, the right hand side of the parallelism can offer any number of interactions over the channels in `Interface`. When `Ticking` finishes waiting, `s` is incremented, and the parallelism synchronises on `tick`.

In this case, the action `Interface` is interrupted and both sides of the parallelism terminate. At this point, the recursive call (on `X`) takes place.

When the parallelism starts, both sides receive a copy of the state, and when the parallelism terminates, the state is updated based on the changes performed by the two sides (on their copies of the state) and the partition of the state. A consequence of this is that changes to the state performed by `Ticking` can only reflect in the behaviour of `Interface` when the parallelism terminates, the state is updated and `Interface` restarts (as part of the recursive call) with a copy of the updated state.

Now we have a clock that not only signals the passing of time, but can also output the time. However, we might also want to be able to restart the clock. For this, we define a channel `restart` and a new clock `RestarableClock`.

```
channels
    restart
```

We define the restartable clock similarly to the process `Clock` defined above. The restartable clock process `RestartableClock` has a new action `Cycle`, and the altered main action offers the action `Cycle` and the possibility of interrupting it through the channel `restart`. If the interruption takes place, the main action recurses and `Cycle` is called resetting the state.

```
process RestartableClock =
begin
    state c: ClockSt
    functions
        get_minute(s: nat) m: nat
        post m = s/60

        get_hour(s: nat) h: nat
        post h = get_minute(s)/60

        get_day(s: nat) d: nat
        post d = get_hour(s)/24
    actions
        Ticking = Wait 1; c.increment(); tick -> Skip
        Interface = second!(c.s) -> Interface
                    [] minute!(get_minute(c.s)) -> Interface
                    [] hour!(get_hour(c.s)) -> Interface
                    [] day!(get_day(c.s)) -> Interface
        Cycle = c.Init(); mu X @ (
                Ticking
                [| {c} | {|tick|} | {} |]
                (Interface/\tick -> Skip)
              ); X
    @ mu X @ Cycle /\ restart -> X
end
```

We can further extend the functionality of the clock by specifying a multi-clock. A simple way of defining such a clock is to compose a number of restartable clocks (or any other variety of clock). This raises the question of how the clocks are composed. For instance, do all clocks synchronise on tick? Can they be restarted on a one by one basis? We present below two processes that model a multi-clock. Both of them assume that the clocks are synchronous, but the first allows independent restarting, while the second does not.

First, we define a number of channels that allow the environment to communicate with specific clocks. We assume that the clocks in the multi-clock are numbered by natural numbers, and are declared in an equivalent way to the ones already defined (except for `tick`), communicating a natural number (the identifier of the clock) and the value originally communicated. We prefix the name of the channels with an `i`.

```
channels
    isecond, iminute, ihour, iday: nat * nat
    irestart: nat
```

Our first model of a multi-clock is specified by the process `NRestartableClocks1`. This is a parameterised process that takes the number `n` of clocks, and starts `n` copies of `RestartableClock` running in parallel and synchronising on `tick`. The channels in the `RestartableClock` process need to be renamed, otherwise we would not be able to distinguish one clock from another. We rename each channel (except `tick`) to its `i` version, communicating the identifier of the clock.

```
process NRestartableClocks1 = n: nat @
    [|{|tick|}|] i: {1,...,n} @
        RestartableClock[[second <- isecond.i,
                          minute <- iminute.i,
                          hour <- ihour.i,
                          day <- iday.i,
                          restart <- irestart.i]]
```

Our alternative process NRestartableClocks2 is similar, except that the different clocks synchronise on restart as well, and this channel is not renamed. Thus, a synchronisation on restart restarts all the clocks simultaneously.

```
process NRestartableClocks2 = n: nat @
    [|{|tick, restart|}|] i: {1,...,n} @
        RestartableClock[[second <- isecond.i,
                          minute <- iminute.i,
                          hour <- ihour.i,
                          day <- iday.i]]
```

One might consider that, whilst these definitions are reasonably intuitive, they are not the most efficient for implementation purposes. So, one might implement a multi-clock simply by associating each channel of a restartable clock with the equivalent i channel, but ranging over all the possible clocks. The next process models such an solution.

```
process NRestartableClocksImpl = n: nat @
    RestartableClock[[second <- isecond.i,
                      minute <- iminute.i,
                      hour <- ihour.i,
                      day <- iday.i | i in set {1,...,n}]]
```

This process simply renames each channel of RestartableClock (except tick and restart) to a set of communications on the associated i channel communicating the identifiers of the clocks.

This process raises the question of which of our multi-clock processes is being implemented by NRestartableClocksImpl. This questions can be formulated as follows.

```
assert NRestartableClocks1 [= NRestartableClocksImpl
assert NRestartableClocks2 [= NRestartableClocksImpl
```

The first assertion states that NRestartableClocksImpl is a refinement of NRestartableClocks1, and the second asserts that the implementation is a refinement of NRestartableClocks2. For some models, this assertions can be checked using a model-checker, but for other, a theorem-prover may be necessary. The CML tools will help answer such questions.

2.4 Unifying Theories of Programming

The semantic models of *Circus* and CML are based on Hoare & He's *Unifying Theories of Programming* [HJ98]. The UTP is a framework in which the theory of relations is used as a unifying basis for programming science across many different computational paradigms: procedural and declarative, sequential and parallel, closely-coupled and distributed, and hardware and software. All programs, designs, and specifications are interpreted as relations between an initial observation and a single subsequent observation, which may be either an intermediate or a final observation, of the behaviour of program execution.

Common ideas, such as sequential composition, conditional, nondeterminism, and parallel composition are shared by different theories of different programming paradigms. For instance, sequential composition is relational composition, conditional is boolean connective, nondeterminism is disjunction, and parallel composition is a restricted form of conjunction. Miracle is interpreted as an empty relation, abortion is interpreted as the universal relation, and correctness and refinement is interpreted as inclusion of relations: reverse implication. All the laws of the relational calculus may be used for reasoning about correctness in all theories and in all languages.

Three elements of a theory are used to differentiate different programming languages and design calculi: the alphabet, a set of names that characterise a range of external observations of a program behaviour; the signature, which provides syntax for denoting the objects of the theory; and the healthiness conditions, which select the objects of a sub-theory from those of a more expressive theory in which it is embedded.

The alphabet of a theory collects the names within the theory that identify observation variables that are important to describe all relevant aspects of a program behaviour. The initial observations of each of these variables are undecorated and compose the input alphabet ($in\alpha$) of a relation. Subsequent observations are decorated with a dash and compose the output alphabet ($out\alpha$) of a relation. This allows a relation to be expressed as in Z by its characteristic predicate. Table 1 summarises the observational variables of the UTP that are used in the semantics of *Circus*.

In *Circus*, some combinations of these variables have interesting semantic meaning. For instance, $okay' \land wait'$ represents a non-divergent state of a process that is waiting for some interaction with the environment; if, however, we have $okay' \land \neg\ wait'$, the non-divergent process has terminated; finally,

okay	This boolean variable indicates if the system has been properly started in a stable state, in which case its value is *true*, or not; $okay'$ means subsequent stabilisation in an observable state.
tr	This variable, whose type is a sequence of events, records all the events in which a program has engaged.
wait	This boolean variable distinguishes the intermediate observations of waiting states from final observations on termination. In a stable intermediate state, $wait'$ has *true* as its value; a *false* value for $wait'$ indicates that the program has reached a final state.
ref	This variable describes the responsiveness properties of the process; its type is a set of events. All the events that may be refused by a process before the program has started are elements of *ref*, and possibly refused events at a later moment are referred by ref'.
v	All program variables (state components, input and local variables, and parameters) are collectively denoted by v.

Table 1: *Circus* Alphabet

$\neg\ okay'$ represents a divergent process.

Besides these variables, there are also UTP theories that include variables that may be used to represent program control, real time clock, or resource availability. For each theory, we may select a subset of relevant variables.

The signature of a theory is a set of operators and atomic components of this theory: it is the syntax of the language. The smaller the signature, the simpler the proof techniques to be applied for reasoning. Signatures may vary according to the theory's purpose. Specification languages are least restrictive and often include quantifiers and all relational calculus operators. Design languages successively remove non-implementable operators. The negation is the first one to be removed. Thus, all operators are monotonic, and recursion can safely be introduced as a fixed-point operator. Finally, programming languages present only implementable operators in their signature. They are commonly defined in terms of their observable effects using the more general specification language.

Healthiness conditions are used to test a specification or design for feasi-

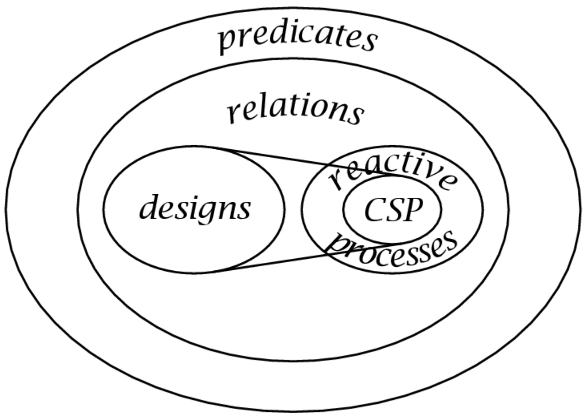


Figure 4: Theories in the UTP

bility, and reject it if it makes implementation impossible in the target language. They are expressed in terms of an idempotent function ϕ that makes a program healthy. Every healthy program P must be a fixed-point $P = \phi(P)$. Some healthiness conditions are used to identify the set of relations that are designs (**H1** and **H2**), reactive processes (**R1**-**R3**), and CSP processes (**CSP1**-**CSP2**).

In Figure 4, we present how some of the theories presented in [HJ98] are related. Relations are predicates with an input and an output (dashed) alphabet. Designs are relations that are **H1** and **H2** healthy. Reactive processes are **R1**, **R2** and **R3** healthy relations (this composition is represented by the healthiness condition **R**). Finally, there are two ways of characterising the CSP processes: they are characterised as reactive processes that are **CSP1** and **CSP2** healthy, or as relations that result from applying **R** to designs.

3 Systematic Development of Trustworthy Component Systems

In this section, we discuss the theoretical background of the report. We present the basic definitions and the composition rules in Section 3.1. The extended counterparts of the definitions and the composition rules are presented in Section 3.2. A full account on the theoretical background can be found elsewhere [Ram11].

Sections 3.3 and 3.4 present novel results. The former defines CSP assertions for all side conditions of the composition rules and the latter discusses the experiment we did to demonstrate the practical effectiveness of the approach by comparing the costs of the verification of the side conditions imposed by the composition rules with the costs of an *ad hoc* verification of the resulting composite system. We also explore variations of the composition rules presented in [RSM09], with the notion of metadata.

3.1 Component Model

Our approach is based on a component model that delimits the broad outline of what constitutes a component, exposing its necessary related technical concepts and constraints. Both, components and connectors, as well as their interaction semantics, are characterised in this component model that defines the building blocks of our systematic development approach. A component contract[1], whose definition is presented below, encapsulates a component in our approach. They are defined in terms of their behaviour (represented as a CSP process), ports (represented as channels) and respective interfaces (types).

Definition 3.1 (Component contract) *A component contract Ctr comprises an observational behaviour $\mathcal{B}$, a set of communication channels $\mathcal{C}$, a set of interfaces $\mathcal{I}$, and a* **total** *function $\mathcal{R} : \mathcal{C} \to \mathcal{I}$ between channels and interfaces of the contract ($Ctr : \langle \mathcal{B}, \mathcal{R}, \mathcal{I}, \mathcal{C} \rangle$), such that:*

- *$\mathcal{B}$ is an I/O process as defined below;*
- *Let $c \in \mathcal{C}$:*

[1]In the COMPASS project contracts are described in CML. Here, a contract is a tuple that includes a behavioural specification (originally described in CSP, but lifted here to CML), and other elements that describe the ports and their types.

- $outputs(c, \mathcal{B}) = \{out.x : \mathcal{R}(c) \bullet c.out.x\}$, *and;*
- $inputs(c, \mathcal{B}) = \{in.x : \mathcal{R}(c) \bullet c.in.x\}$

Intuitively, the component $\mathcal{R}$ describes the component's channels and their respective types.

Our approach follows approaches like that of [All97b] in which component models have a higher-level granularity by complementing the syntactical information of a component with behaviour. In our case, we explicitly separated inputs and outputs.

The behaviour of these components are represented by I/O processes, which are CSP processes P that satisfy five conditions, which are formally presented in [Ram11]:

- **I/O Channels.** Every channel in P is either an input channel or an output channel. Formally, we say a channel c is an I/O channel if there exists two functions, $inputs(c, P)$ and $outputs(c, P)$, for every process P, such that:
  - $inputs(c, P) \cup outputs(c, P) \subseteq \{\!| \ c \ |\!\}$, and
  - $inputs(c, P) \cap outputs(c, P) = \emptyset$.

  Formally, the two functions map pairs $CHANNEL \times PROCESS$ to a set of events. In Appendix H, we present a full type-checked Z formalisation of our compositional model.
- **Infinite Traces.** P has an infinite set of traces (but finite state-space);
- **Divergence-freedom.** P is divergence-free;
- **Input Determinism.** If a set of input events in P are offered to the environment, none of them are refused. Formally, we say a process P is input deterministic if:
  - $\forall s \frown \langle c.a \rangle : traces(P) \mid \ c.a \in inputs(c, P) \bullet$ $(s, \{c.a\}) \notin failures(P)$
- **Strong Output Decisive.** All choices (if any) among output events on a given channel in P are internal. The process, however, must offer at least one output on that channel. Formally, we say a process is strong output decisive if:
  - $\forall s \frown \langle c.b \rangle : traces(P) \mid c.b \in outputs(c, P) \bullet$ $(s, outputs(c, P)) \notin failures(P)$ $\land (s, outputs(c, P) \setminus \{c.b\}) \in failures(P)$

These conditions lay the foundations of our composition rules for contracts whenever every two components are compatible to interoperate. The application of the composition rules and the characterisation constraints in the component model impose side conditions that, if satisfied, ensure deadlock freedom in the composition result. Hence, in our approach, problems are anticipated before all parts are integrated.

In [Ram11], we present four composition rules; each one focuses on a specific scenario at composition. The rules provide asynchronous pairwise compositions and focus on the preservation of deadlock freedom in the resulting component. The preservation of livelock-freedom is not in the scope of this report but also discussed in [Ram11]. Using the rules, developers may synchronise two channels of two components, or even of the same component. The four rules are *interleave*, *communication*, *feedback* and *reflexive* compositions. The first three rules have also been presented in [RSM09].

The interleave composition rule is the simplest form of composition. It aggregates two independent entities such that, after composition, these entities still do not communicate between themselves. They directly communicate with the environment as before, with no interference from each other. The only proviso states that they do not share any communication channel.

Definition 3.2 (Interleave composition) *Let P and Q be two component contracts, such that:*

- *P and Q have disjoint channels, and;*
- *$\mathcal{C}_P \cap \mathcal{C}_Q = \emptyset$.*

Then, the interleave composition of P and Q (namely $P\,[|||]\,Q$) is given by:

$$P\,[|||]\,Q = P_{\langle\rangle} \asymp {}_{\langle\rangle}Q$$

This definition and others that follow use the direct composition operator $\asymp$, which provides an asynchronous interaction, mediated by infinite buffers, between corresponding channels from two lists. In this rule, no channel participates in the operation.

The result of an application of a composition rule is a new component. In many cases, it is necessary to provide a means to connect to two channels of a same component. This, however, is not possible using CSP as it does not provide any constructor for a reflexive direct connection. For that, we use a buffered communication as means to permit such reflexive communication in a component. By providing an asynchronous interaction, we also offer a more generic approach that allows its use in both asynchronous and synchronous systems. On the other hand, the costs of verification are knowingly higher for buffered asynchronous specifications. This cost, however, is alleviated by the use of metadata as we discuss in Section 3.2.

The next composition rule needs the properties below.

Prop. i. (**I/O Confluence**) Whenever a state has two alternative actions α and β, then performing either of them does not preclude the other, unless it is a choice among inputs or outputs of the same channel;

Prop. ii. (**Finite Output Property**) They always communicate a finite number of outputs. As I/O processes are divergence-free, the absence of divergence after hiding the outputs in the original protocol guarantees this property.

Prop. iii. (**Strong Compatibility**) There must always be an output event to be performed, and at least one of the processes must have all enabled outputs accepted by the other process.

The first two properties deal with buffering concerns in order to allow mechanical verifications on the system without state explosion [Ros05]. The third property guarantees the interoperability of the two components. Here, the formal definitions are omitted for the sake of conciseness. They can be found in [Ram11].

The second composition rule states the most common way for linking complementary channels of two different entities.

Definition 3.3 (Communication composition) *Let P and Q be two component contracts, and ic and oc two communication channels, such that:*

- $ic \in \mathcal{C}_P \wedge oc \in \mathcal{C}_Q$*;*
- $\mathcal{C}_P \cap \mathcal{C}_Q = \emptyset$*, and;*
- *the port protocols* $Prot_{IMP}(P, ic) [\![R_{IO}^{ic \to oc}]\!]$ *and* $Prot_{IMP}(Q, oc) [\![R_{IO}^{oc \to ic}]\!]$ *are I/O confluent strong compatible and satisfy the finite output property.*

Then, the communication composition of P and Q (namely $P[ic \leftrightarrow oc]Q$) via ic and oc is defined as follows:

$$P[ic \leftrightarrow oc]Q = P_{\langle ic \rangle} \asymp_{\langle oc \rangle} Q$$

Besides having disjoint channel sets, further restrictions apply to the divergent-free processes implementation protocols on the linked channels ($Prot_{IMP}$), which are given by the abstraction of their behaviour projection over these channels. These restrictions, however, apply to a renamed version of these protocols: $[\![R_{IO}^{oc \to ic}]\!]$ replaces outputs of oc by inputs of ic.

Practical developments also present more complex systems with cycles of dependencies in the topology of the system structure; undesirable cycles need to be avoided. The feedback composition provides the possibility of creating safe cycles.

Definition 3.4 (Feedback composition) *Let P be a component contract, and ic and oc two communication channels, such that:*

- *the protocols $Prot_{IMP}(P, ic)\, [\![R_{IO}^{ic \to oc}]\!]$ and $Prot_{IMP}(P, oc)\, [\![R_{IO}^{oc \to ic}]\!]$ are I/O confluent strong compatible and satisfy the finite output property, and;*
- *$\{ic, oc\} \subseteq \mathcal{C}_P$ and decoupled in P.*

Then, the feedback composition P ($P[oc \hookrightarrow ic]$) hooking oc to ic is defined as follows:

$$P[oc \hookrightarrow ic] = P \asymp|^{\langle ic \rangle}_{\langle oc \rangle}$$

This rule imposes some conditions that are similar to those in the communication composition rule (relative to protocol compatibility and buffer tolerance), except that it additionally imposes that channels are decoupled.

Prop. iv. (**Decoupled Channels**) Communication on one channel does not interfere on communications through the other (their communications are interleaved). Formally, the channels within Ch are decoupled in P if, and only, if $P \upharpoonright Ch \equiv_{\mathrm{F}} \interleave_{z \in Ch} Prot_{IMP}(P, z)$.

The composition rules presented so far deal with systems with a tree topology. In practice, there are more complex systems that indeed present cycles of dependencies in the topology of the system structure. The last composition rule, *reflexive composition*, is more general than the feedback one. However, it is also more costly regarding verification.

Definition 3.5 (Reflexive composition) *Let P be a component contract, and ic and oc two communication channels, such that:*

- *$\{ic, oc\} \subseteq \mathcal{C}_P$, and;*

- *$P \upharpoonright \{ic, oc\}$ is buffering self-injection compatible and satisfies the finite output property.*

Then, the reflexive composition P (namely $P[oc \hookrightarrow ic]$) hooking oc to ic is defined as follows:

$$P[ic \hookrightarrow oc] = P \asymp|^{\langle ic \rangle}_{\langle oc \rangle}$$

This rule requires that the projection on the two linked channels ($P \upharpoonright \{ic, oc\}$) satisfies the finite output property and the projection is buffering self-injection compatible.

Prop. v. (**Buffering Self-injection Compatibility**) allows the injection of information from one channel to the other via the implicit buffers of the composition. Formally, a buffering self-injection compatible process can establish a communication between its channels via a one-place buffer without deadlock.

From our proposed building block constructors (composition rules), any system S can be structured as follows.

$$S ::= P \mid S\,[|||]\,S \mid S[c_1 \leftrightarrow c_2]S \mid S[c_1 \hookrightarrow c_2] \mid S[c_1 \hookrightarrow c_2]$$

where P is a component contract whose behaviour is deadlock free. We say that any component system that follows this grammar is in *normal form*.

The following theorem from [Ram11] guarantees that components arising from the application of the rules to deadlock-free components are also deadlock-free.

Theorem 3.1 *(Deadlock-free Component Systems) Any system S in normal form, built from deadlock-free components, is deadlock-free.*

In addition to the contract elements previously presented, we may also define an enriched component contract (*BRICK-components*). These components enrich the original contracts with metadata that record by construction information that can be used to alleviate some verification conditions during component composition. This enriched components contract, the corresponding composition rules and the mechanisation of the composition rules side conditions in CSP is presented in the next section.

3.2 Extended Component Model

In our approach, metadata comprise information that can (at any moment) be derived from other component contract elements. Such metadata enriches component contracts with static information that assists the runtime environment with additional (validation) properties. The metadata information is: (1) dual protocols; (2) context protocols; (3) protocol implementations; and (4) decoupled channels. Informally, the behaviour of the dual protocol of a process P after a trace s is always an external choice of the outputs and one of the inputs of P, if it exists, after s. Furthermore, a context protocol of a process P is a deadlock-free deterministic process that has the same traces as P. Both are used in protocol compatibility verifications. The main metadata information selected in our approach are decoupled channels and protocol implementations. These are important conditions in the communication and feedback compositions rules. Similarly to the composition rules presented before, we presented four composition rules for enriched component contracts. In particular, we use metadata to alleviate several verifications in our rigorous strategy for component compositions. The extended contracts specialise the notion of *protocol oriented* component and enrich their contract with metadata.

Definition 3.6 (Enriched component contract) *Let Ctr be a protocol oriented component contract, and $\mathcal{K}$ a metadata derived from its elements. An enriched component contract that includes Ctr is represented by:*

$$\langle \mathcal{B}_{Ctr}, \mathcal{R}_{Ctr}, \mathcal{I}_{Ctr}, \mathcal{C}_{Ctr}, \mathcal{K} \rangle$$

where $\mathcal{K}$ comprises the following information:

$$\mathcal{K} : \langle Prot^{\mathcal{K}},\ CTX^{\mathcal{K}},\ DProt^{\mathcal{K}},\ Dec^{\mathcal{K}} \rangle$$

such that:

- $\mathrm{dom}\ Prot^{\mathcal{K}} \subseteq \mathcal{C}_{Ctr}\ \wedge$
  $\forall\, c : \mathrm{dom}\ Prot^{\mathcal{K}} \bullet Prot^{\mathcal{K}}(c) \sqsubseteq_{\mathrm{F}} Prot_{IMP}(Ctr, c)$
- $\mathrm{dom}\ DProt^{\mathcal{K}} \subseteq \mathcal{C}_{Ctr}\ \wedge$
  $\forall\, c : \mathrm{dom}\ DProt^{\mathcal{K}} \bullet DProt^{\mathcal{K}}(c)$ *is the dual protocol of* $Prot^{\mathcal{K}}(c)$
- $\mathrm{dom}\ CTX^{\mathcal{K}} \subseteq \mathcal{C}_{Ctr}\ \wedge$
  $\forall\, c : \mathrm{dom}\ CTX^{\mathcal{K}} \bullet CTX^{\mathcal{K}}(c)$ *is the context process of* $Prot^{\mathcal{K}}(c)$
- $\mathrm{dom}\ Dec^{\mathcal{K}} \subseteq \mathcal{C}_{Ctr}\ \wedge\ \mathrm{ran}\ Dec^{\mathcal{K}} \subseteq \mathcal{C}_{Ctr}$
- $\forall\, c_1, c_2 : \mathcal{C}_{Ctr} \bullet c_1\ Dec^{\mathcal{K}}\ c_2 \Rightarrow \{c_1, c_2\}\ DecoupledIn\ Ctr\ \wedge\ c_2\ Dec^{\mathcal{K}}\ c_1$

The element $Prot^{\mathcal{K}}$ is a relation from channels to protocols, which represent the actual port-protocol of the component on that channel. If a protocol within $Prot^{\mathcal{K}}$ satisfies a property, then, by refinement, it also holds for the protocol of the component. Similarly, the elements $DProt^{\mathcal{K}}$ and $CTX^{\mathcal{K}}$ map channels into context processes and dual protocols, respectively. They are used to support the use of the protocols within $Prot^{\mathcal{K}}$; these are used, for instance, in protocol compatibility verifications. Finally the element $Dec^{\mathcal{K}}$ is a relation among decoupled channels of the component.

Since these metadata comprise derived information, it can be ignored by a composition environment, and, furthermore, the component can still be used in environments unaware of them. As a consequence, despite the use of metadata can be considered a powerful tool during the integration phase, its use is optional.

To increase the value of our compositional approach, we derive composition metadata from the metadata of the original components, without always building them from scratch. After each composition rule is applied, the metadata are updated using simple formulae that consider the semantics of such composition rule.

Similarly to the composition rules presented before, we present four composition rules for enriched component contracts. In order to preserve protocols behaviours after each composition and to store them in metadata, enriched components require a stronger verification of protocol compatibility, which we call matching compatible.

Similarly to the rules presented before, we present four new composition rules for enriched component contracts. In order to preserve protocol behaviours after each composition and to store them in metadata, the new rules require a stronger notion of protocol compatibility, which we call matching compatibility.

Prop. vi. (**Matching Compatibility**) Two protocols R and S are compatible if the dual protocol of R is failure equivalent to S. Formally, two port-protocols P and Q are matching compatible if, and only if, $DProt(P) \equiv_{\mathrm{F}} Q$.

This kind of compatibility is subtly different from strong compatibility. The former is even stronger than the latter. The advantage of compositions in which the protocols are matching compatible is that it preserves local progress and, furthermore, other protocols (not involved in the composition) are preserved.

The simplest composition of enriched component contracts is the one formed by the interleaving of its components.

Definition 3.7 (Enriched interleaving composition) *Let P and Q be two enriched component contracts, such that P and Q have disjoint channels, $\mathcal{C}_P \cap \mathcal{C}_Q = \emptyset$. Then, the enriched interleaving composition of P and Q (namely $P \,[|||]\, Q$) is given by:*

$$\begin{aligned}P \,[|||]^e\, Q = Enrich(&\langle \mathcal{B}_P, \mathcal{R}_P, \mathcal{I}_P, \mathcal{C}_P \rangle_{\langle\rangle} \asymp {}_{\langle\rangle} \langle \mathcal{B}_Q, \mathcal{R}_Q, \mathcal{I}_Q, \mathcal{C}_Q \rangle, \\ &\langle Prot^{\mathcal{K}}_{PQ}, CTX^{\mathcal{K}}_{PQ}, DProt^{\mathcal{K}}_{PQ}, Dec^{\mathcal{K}}_{PQ} \rangle)\end{aligned}$$

where

(i) $Prot^{\mathcal{K}}_{PQ} = Prot^{\mathcal{K}}_P \cup Prot^{\mathcal{K}}_Q$

(ii) $CTX^{\mathcal{K}}_{PQ} = CTX^{\mathcal{K}}_P \cup CTX^{\mathcal{K}}_Q(c)$

(iii) $DProt^{\mathcal{K}}_{PQ} = DProt^{\mathcal{K}}_P \cup DProt^{\mathcal{K}}_Q$

(iv) $Dec^{\mathcal{K}}_{PQ} =$
$Dec^{\mathcal{K}}_P \cup Dec^{\mathcal{K}}_Q \cup \{(c_1, c_2) \mid (c_1 \in \mathcal{C}_Q \wedge c_2 \in \mathcal{C}_P) \vee (c_1 \in \mathcal{C}_P \wedge c_2 \in \mathcal{C}_Q)\}$

The result of this composition is similar to the one from Definition 3.2. In addition, we show here the metadata associated to the interleaving. At this moment, no benefit is obtained from the metadata; they are maintained for more complex compositions. However, the calculation of metadata is very simple. It basically includes all information of the metadata of P and Q, except that it also states that all channels of one component are decoupled from the other; this is a direct result of the interleaved behaviour of the composition.

Similarly, we define communication compositions of enriched component contracts in the following way.

Definition 3.8 (Enriched communication composition) *Let P and Q be two enriched component contracts, and ic and oc two channels, such that:*

- $ic \in \mathcal{C}_P \wedge oc \in \mathcal{C}_Q$;
- $\mathcal{C}_P \cap \mathcal{C}_Q = \emptyset$, *and;*
- *the port protocols* $Prot^{\mathcal{K}}_P(ic) \,\|[\, R^{ic \to oc}_{IO}]\|$ *and* $Prot^{\mathcal{K}}_Q(oc) \,\|[\, R^{oc \to ic}_{IO}]\|$ *are I/O confluent matching compatible and satisfies the finite output property.*

Then, the communication composition $P[ic \leftrightarrow oc]Q$ is defined as follows:

$$P[ic \leftrightarrow oc]^e Q = \\ \quad \mathit{Enrich}\left(\begin{array}{l}\langle \mathcal{B}_P, \mathcal{R}_P, \mathcal{I}_P, \mathcal{C}_P\rangle_{\langle ic\rangle} \asymp {}_{\langle oc\rangle}\langle \mathcal{B}_Q, \mathcal{R}_Q, \mathcal{I}_Q, \mathcal{C}_Q\rangle, \\ \langle \mathit{Prot}^{\mathcal{K}}_{PQ}, \mathit{CTX}^{\mathcal{K}}_{PQ}, \mathit{Dec}^{\mathcal{K}}_{PQ}\rangle\end{array}\right)$$

where

$$\begin{array}{l} \mathit{Prot}^{\mathcal{K}}_{PQ} = \{(c, \mathit{Prot}^{\mathcal{K}}_P(c)) \mid c \in \mathrm{dom}\, \mathit{Prot}^{\mathcal{K}}_P \setminus \{ic\}\} \\ \qquad\qquad \cup \{(c, \mathit{Prot}^{\mathcal{K}}_Q(c)) \mid c \in \mathrm{dom}\, \mathit{Prot}^{\mathcal{K}}_Q \setminus \{oc\}\} \end{array}$$

$$\begin{array}{l} \mathit{DProt}^{\mathcal{K}}_{PQ} = \{(c, \mathit{DProt}^{\mathcal{K}}_P(c)) \mid c \in \mathrm{dom}\, \mathit{DProt}^{\mathcal{K}}_P \setminus \{ic\}\} \\ \qquad\qquad \cup \{(c, \mathit{DProt}^{\mathcal{K}}_Q(c) \mid c \in \mathrm{dom}\, \mathit{DProt}^{\mathcal{K}}_Q \setminus \{oc\}\} \end{array}$$

$$\begin{array}{l} \mathit{CTX}^{\mathcal{K}}_{PQ} = \{(c, \mathit{CTX}^{\mathcal{K}}_P(c)) \mid c \in \mathrm{dom}\, \mathit{CTX}^{\mathcal{K}}_P \setminus \{ic\}\} \\ \qquad\qquad \cup \{(c, \mathit{CTX}^{\mathcal{K}}_Q(c) \mid c \in \mathrm{dom}\, \mathit{CTX}^{\mathcal{K}}_Q \setminus \{oc\}\} \end{array}$$

$$\mathit{Dec}^{\mathcal{K}}_{PQ} = \left\{\begin{array}{l} (c_1, c_2) \mid \{c_1, c_2\} \cap \{ic, oc\} = \emptyset \\ \qquad \wedge \left(\begin{array}{l} \left(\begin{array}{l} (c_1\ \mathit{Dec}^{\mathcal{K}}_P\ ic \vee ic\ \mathit{Dec}^{\mathcal{K}}_P\ c_1) \\ \wedge\, (c_2 \in \mathcal{C}_Q \vee c_1 \mathit{Dec}^{\mathcal{K}}_P c_2) \end{array}\right) \\ \vee \left(\begin{array}{l} (oc\ \mathit{Dec}^{\mathcal{K}}_Q\ c_2 \vee c_2\ \mathit{Dec}^{\mathcal{K}}_Q\ oc) \\ \wedge\, (c_1 \in \mathcal{C}_P \vee c_1 \mathit{Dec}^{\mathcal{K}}_Q c_2) \end{array}\right) \end{array}\right) \end{array}\right\}$$

The result of this composition is similar to the one from Definition 3.3, except for: instead of checking compatibility among port protocols of the original components, we check it on port protocols within their metadata. Furthermore, the composition does not have to take into account the complexity of its components, since no port-protocol has to be derived from the component behaviours. In addition, we show here the metadata associated to the composition, which can be used in further compositions. Again, the calculation of metadata is very simple. They include all information of the metadata of P and Q, excluding information about ic and oc, which does not belong to the new composition contract. There are also new relations identified among channels of one component and channels of the other, requiring that these channels are decoupled with the channels involved in the composition (ic and oc). This results from the semantics of the parallel operator being used in the composition. Observe that $\mathit{Dec}^{\mathcal{K}}$ is a symmetric relation, and, furthermore, this has to be handled in its calculation.

Now we define the feedback composition of an enriched component contract.

Definition 3.9 (Enriched feedback composition) *Let P be an enriched component contract, and ic and oc two communication channels, such that:*

- *$\{ic, oc\} \subseteq \mathcal{C}_P$;*
- *the port protocols $Prot_P^{\mathcal{K}}(ic) \,[\![\, R_{IO}^{ic \to oc}]\!]$ and $Prot_P^{\mathcal{K}}(oc) \,[\![\, R_{IO}^{oc \to ic}]\!]$ are I/O confluent matching compatible and satisfies the finite output property, and;*
- *$ic\ Dec_P^{\mathcal{K}}\ oc$.*

Then, the feedback composition P (namely $P[oc \hookrightarrow ic]$) hooking oc to ic is defined as follows:

$$\begin{aligned} P[oc \hookrightarrow ic]^e = Enrich(&\langle \mathcal{B}_P, \mathcal{R}_P, \mathcal{I}_P, \mathcal{C}_P \rangle \asymp|_{\langle oc \rangle}^{\langle ic \rangle}, \\ &\langle Prot_S^{\mathcal{K}}, CTX_S^{\mathcal{K}}, DProt_S^{\mathcal{K}}, Dec_S^{\mathcal{K}} \rangle) \end{aligned}$$

where

$$Prot_{PQ}^{\mathcal{K}} = \{(c, Prot_P^{\mathcal{K}}(c)) \mid c \in \operatorname{dom} Prot_P^{\mathcal{K}} \setminus \{ic, oc\}\}$$
$$DProt_{PQ}^{\mathcal{K}} = \{(c, DProt_P^{\mathcal{K}}(c)) \mid c \in \operatorname{dom} DProt_P^{\mathcal{K}} \setminus \{ic, oc\}\}$$
$$CTX_{PQ}^{\mathcal{K}} = \{(c, CTX_P^{\mathcal{K}}(c)) \mid c \in \operatorname{dom} CTX_P^{\mathcal{K}} \setminus \{ic, oc\}\}$$
$$Dec_{PQ}^{\mathcal{K}} = \left\{ \begin{array}{l} (c_1, c_2) \mid \{c_1, c_2\} \cap \{ic, oc\} = \emptyset \\ \qquad \wedge\ c_1\ Dec_P^{\mathcal{K}}\ c_2 \\ \qquad \wedge \left(\begin{array}{l} (c_1\ Dec_P^{\mathcal{K}}\ ic \wedge c_1\ Dec_P^{\mathcal{K}}\ oc) \\ \vee\ (ic\ Dec_P^{\mathcal{K}}\ c_2 \wedge oc\ Dec_P^{\mathcal{K}}\ c_2) \end{array} \right) \end{array} \right\}$$

The result of this composition is similar to the one from Definition 3.4, except that most provisos use the metadata of its original components directly. Instead of having to check compatibility among port protocols of P, we check this on port protocols within the metadata. Instead of verifying that two channels are decoupled in P, we verify it directly on relations within the metadata. In this way, we perform lightweight verifications. Moreover, the composition does not have to take into account the complexity of P. In addition, we show here the metadata associated to the composition, which can be used in further compositions. Again, the calculation of metadata is very simple. The new metadata include all information of the metadata of P, excluding information about ic and oc, which does not belong to the composition contract. Some other channels are also removed from the decoupled relation $Dec_S^{\mathcal{K}}$, since after the composition new communications are established.

The last rule is the reflexive composition of enriched compositions.

Definition 3.10 (Enriched reflexive composition) *Let P be a component contract, and ic and oc two communication channels, such that:*

- $\{ic, oc\} \subseteq \mathcal{C}_P$*, and;*
- $P \upharpoonright \{ic, oc\}$ *is buffering self-injection compatible and satisfies the finite output property.*

Then, the reflexive composition P *(namely* $P[oc \hookrightarrow ic]$*) hooking* oc *to* ic *is defined as follows:*

$$\begin{aligned} P[ic \hookrightarrow oc]^e = \mathit{Enrich}(&\langle \mathcal{B}_P, \mathcal{R}_P, \mathcal{I}_P, \mathcal{C}_P \rangle \asymp|^{\langle ic \rangle}_{\langle oc \rangle}, \\ &\langle \mathit{Prot}^{\mathcal{K}}_S, \mathit{CTX}^{\mathcal{K}}_S, \mathit{DProt}^{\mathcal{K}}_S, \mathit{Dec}^{\mathcal{K}}_S \rangle) \end{aligned}$$

where

$$\mathit{Prot}^{\mathcal{K}}_{PQ} = \{(c, \mathit{Prot}^{\mathcal{K}}_P(c)) \mid c \in \operatorname{dom} \mathit{Prot}^{\mathcal{K}}_P \setminus \{ic, oc\}\}$$
$$\mathit{DProt}^{\mathcal{K}}_{PQ} = \{(c, \mathit{DProt}^{\mathcal{K}}_P(c)) \mid c \in \operatorname{dom} \mathit{DProt}^{\mathcal{K}}_P \setminus \{ic, oc\}\}$$
$$\mathit{CTX}^{\mathcal{K}}_{PQ} = \{(c, \mathit{CTX}^{\mathcal{K}}_P(c)) \mid c \in \operatorname{dom} \mathit{CTX}^{\mathcal{K}}_P \setminus \{ic, oc\}\}$$
$$\mathit{Dec}^{\mathcal{K}}_{PQ} = \left\{ \begin{array}{l} (c_1, c_2) \mid \{c_1, c_2\} \cap \{ic, oc\} = \emptyset \\ \qquad \wedge\ c_1\ \mathit{Dec}^{\mathcal{K}}_P\ c_2 \\ \qquad \wedge \left(\begin{array}{l} (c_1\ \mathit{Dec}^{\mathcal{K}}_P\ ic \wedge c_1\ \mathit{Dec}^{\mathcal{K}}_P\ oc) \\ \vee\ (ic\ \mathit{Dec}^{\mathcal{K}}_P\ c_2 \wedge oc\ \mathit{Dec}^{\mathcal{K}}_P\ c_2) \end{array} \right) \end{array} \right\}$$

The result of this composition is similar to the one from Definition 3.5. It does not benefit from the metadata of its original components. This is because to check *buffering self-injection compatibility* we cannot solely use port protocols, but the entire component behaviour; it checks the behaviour concerning two communication channels. In addition, we show here the metadata associated to the composition, which can be used in further compositions. The structure of the metadata is identical to the one of a feedback composition of enriched components, since both are unary compositions.

In [Ram11], we provide proofs that guarantee that the result of the application of the extended composition rules are themselves extended component contracts. Observe that all rules presented here also guarantee deadlock freedom because the behaviour of their compositions is equivalent to the behaviour of the general rules used to create them.

3.3 Mechanising the Composition Rules Side Conditions in CSP

In [Ram11], we present a formalisation of all side conditions using a mathematical notation. Their general mechanical verification requires an inte-

Alphabets	`assert STOP [T= RUN(inter(events(P),events(Q)))`

I/O Channels	`assert not`
	`  Test(inter(inputs(P),outputs(P)) == {})`
	`  [T= ERROR`

Infinite Traces	`assert not HideAll(P):[divergence free [FD]]`
Divergence Free	`assert P:[divergence free [FD]]`
Input Determinism	`assert LHS_InputDet(P) [F= RHS_InputDet(P)`

Strong Output Decisive	`assert LHS_OutputDec_A(P)`
	`       [F= RHS_OutputDec_A(P)`
	`assert LHS_OutputDec_B(P,c1)`
	`       [F= RHS_OutputDec_B(P,c1)`
	`assert LHS_OutputDec_B(P,c2)`
	`       [F= RHS_OutputDec_B(P,c2)`

Table 2: Mechanisation of Side Conditions in CSP for Interleave Composition

gration of their encoding in a theorem-prover that supports CSP like CSP-Prover [IR08]. Nevertheless, for a specific case like our case study presented in Section 6, it is possible to define CSP assertions for all these conditions.

By way of illustration, Table 2 presents some of the mechanisation of the side conditions in CSP for the interleave composition of two processes P ($\alpha P = \{c1, c2\}$) and Q described in the sections that follows. Counterparts of the assertions presented in these sections are also needed for process Q. A summary of all mechanisation of the composition rules side conditions in CSP is presented in Appendix F.

3.3.1 Alphabets

The first assertion guarantees that the channels of the processes are disjoint by checking that offering (`RUN`) all events of the intersection (`inter`) between both processes events is a refinement of `STOP`. Since `STOP` offers no events, this is only possible if the intersection is empty.

3.3.2 I/O Channels

The assertion related to I/O channels is similar but is characterised in a different manner because functions *inputs* and *outputs* return channels, not events, and hence cannot be used in `RUN`. Its characterisation test uses two auxiliary processes: `ERROR = error -> SKIP` and `Test(c) = not c & ERROR`. This assertion is only satisfied if the condition is true.

3.3.3 Infinite Traces and Divergence-Freedom

Infinite traces are checked by asserting that hiding all events (`HideAll`) introduces divergence. Both, this check and the one that checks if the process itself is divergence-free are achieved using FDR's built-in divergence check.

The next two assertions proved to be harder than usual and deserve special attention. In [ORS+12a], we present an exercise in the definition of the CSP assertions that characterise input determinism (Section 3.3.4) and strong output decisiveness (Section 3.3.5). In what follows, we present the details of the results achieved in [ORS+12a].

3.3.4 Input Determinism

In [RSM09], we formally define input determinism as follows:

Definition 3.11 (Input determinism) *We say a process P is input deterministic if*

$$\forall\, s \frown \langle c.a \rangle : traces(P) \mid \ c.a \in inputs(c,P) \bullet (s, \{c.a\}) \notin failures(P)$$

Informally, this means that if a set of input events in P are offered to the environment, none of them are refused. As a consequence, the process is defined to be deterministic on the inputs.

In [Ros10], Roscoe presents a refinement check for **divergence-free processes** in FDR that is based on Lazić's Algorithm [Laz99].

The approach is to run two copies of the process synchronising on a newly introduced special event `clunk`. Furthermore, the set `AllButClunk` includes all events that `P` uses, but not the special event `clunk`.

```
channel clunk
AllButClunk = diff(Events,{clunk})
```

This special event is used to synchronise both copies of the process after any event. First, we enforce that the process synchronises in this special event after any other events. This is achieved by running the process is parallel with a watchdog process that produces a `clunk` after any event as follows.

```
Clunking(P) = P [| AllButClunk |] Clunker
Clunker = [] x:AllButClunk @ x -> clunk -> Clunker
```

`Clunking(P)` behaves exactly like `P`, except that it communicates `clunk` between each pair of other events.

Next, we run both controlled copies of the process in parallel, but synchronising only on `clunk`. It follows that

```
(Clunking(P) [|{clunk}|] Clunking(P))\{clunk}
```

allows both copies of `P` to proceed independently, except that their individual traces never differ in length by more than one.

If `P` is deterministic, then, whenever one copy of `P` performs an event, the other one cannot refuse it provided they have both performed the same trace to date. It follows that if we run

```
RHS_InputDet(P) =
    (Clunking(P)[|{clunk}|]Clunking(P)) \ {clunk}
    [|AllButClunk|]
    Repeat

Repeat = [] x:AllButClunk @ x -> x -> Repeat
```

then the result will never deadlock after a trace with odd length. Such a deadlock can only occur if, after some trace of the form `<a,a,...,d,d>` in which each `P` has performed `<a,...,d>`, one copy of `P` accepts some event `e` and the other refuses it. This exactly corresponds to `P` not being $\mathcal{F}$-deterministic.

We can thus check determinism and $\mathcal{F}$-determinism by testing whether the process `RHS_InputDet(P)` refines the following process over $\mathcal{F}$.

```
Deterministic(S) =
    STOP
    |~|
    ([] x:AllButClunk @
        x -> (if member(x,S)
        then x -> Deterministic(S)
        else (STOP |~| x -> Deterministic(S))))

LHS_InputDet(P) = Deterministic(inputs(P))

assert LHS_InputDet(P) [F= RHS_InputDet(P)
```

The process `LHS_InputDet(P)` specifies a deterministic behaviour of the set of input events of a given process (`inputs(P)`). Notice that using `AllButClunk` bring us back to the original Lazićs algorithm, in which

```
LHS_InputDet =
    STOP |~| ([] x:AllButClunk @ x -> x -> LHS_InputDet)
```

that checks determinism in all events. We are, however, interested in a particular set of events `S`, namely the inputs.

Because it runs `P` in parallel with itself, Lazićs algorithm is at worst quadratic in the state space of `P`. (In other words, the number of states can be as many as the square of the state space of `P`.) In most cases, however, it is much better than this, but not as efficient as the FDR check.

Lazić algorithm works (in the respective models) to determine whether a process is deterministic over $\mathcal{FD}$ or $\mathcal{F}$.

The fact that this algorithm is implemented by the user in terms of refinement checking means that it is easy to vary, and in fact many variations on this check have been used when one wants to compare the different ways in which a process `P` can behave on the same or similar traces. We use this idea for Strong Output Decisiveness as we explain in the sequel.

3.3.5 Strong Output Decisiveness

In [RSM09], we formally define Strong Output Decisiveness as follows:

Definition 3.12 (Strong output decisiveness) *We say a process P is strong output decisive if:*

$$\begin{array}{l} \forall\, s \frown \langle c.b\rangle : traces(P) \mid c.b \in outputs(c,P) \bullet \\ \quad (s, outputs(c,P)) \notin failures(P) \\ \quad \wedge\ (s, outputs(c,P) \setminus \{c.b\}) \in failures(P) \end{array}$$

Informally, this means that all choices (if any) among output events on a given channel in P are internal. The process, however, must offer at least one output on that channel. Hence, the choice between output channels is external.

In [Ros05], processes are output decisive on channel `c` if every maximal refusal of the process omits at most one member of `{|c|}`. This definition, however, differs from ours [RSM09] in three main aspects:

1. **Channel based definition**: In [Ros05], we present a channel-based approach. We consider channels are unidirectional for each process; hence, within each process, a channel is either input or output. For this reason, in [Ros05], processes are not allowed to offer an external choice between an input and an output on the same channel as they are here.
2. **Single event outputs**: In [Ros05], process `P = c4.out -> PN` is not strong output decisive. In our definition, though, it is.
3. **Refusing all outputs**: In [Ros05], a strong output decisive process might refuse all outputs on a given channel at once. In [RSM09], we reject such processes as strong output decisive; if a process might offer an output on `c1`, it might not refuse all outputs on `c1` at once. So, both process below are Strong Output Decisive according to [Ros05], but they are not according to [RSM09].

We may, therefore, state the notion of Strong Output Decisiveness used in [RSM09] as follows: a process `P` is Strong Output Decisive if choices between outputs on different channels are external and choices between outputs on the same channel are internal.

The characterisation of strong output decisiveness as assertions will be divided into two parts:

1. The first part, **Part A**, verifies that after a trace `s^<c.x>`, the process cannot refuse all events on `{|c|}`. This verification, however, does not guarantee that choices are non-deterministic.
2. The second part, **Part B**, verifies that trace `s^<c.x>`, the process might refuse all events on `{|c|} \ {c.x}`. Hence, the process is non-deterministic for outputs on that channel.

Part A - Inter-channel Determinism. Let `GET_CHANNELS(P)` be a set of distinct channels used in process `P`. Using the same `Clunker(p)` as previously described in Section 3.3.4, we now use two copies of the clunking version of `P` synchronising on `clunk` and everything except members of the channels we are worrying about, the outputs of `P`.

```
(Clunking(P)[|diff(Events, outputs(P))|]Clunking(P)) \ {clunk}
```

Furthermore, we consider a process `One2Many(S)`, which simply repeats events that are not communication on the channels in `S`; otherwise, it offers any other communication on that channel.

```
One2Many(S) =
    ([] x:diff(Events,union(S,{clunk})) @ x -> One2Many(S))
    [] ([] c:S @ [] x:{|c|} @ x -> One2Many'(S,c,x))

One2Many'(S,c,x) =
    [] y:{|c|} @ y -> if x==y then One2Many(S) else STOP
```

We put this process in parallel with the above to get the right-hand side implementation of the assertion.

```
RHS_OutputDec_A(P) =
    (Clunking(P)[|diff(Events, outputs(P))|]Clunking(P))\{clunk}
    [| AllButClunk |]
    One2Many(outputs(P))
```

This process expects the second copy of P to respond with a member of the same channel when an output has occurred. Importantly, it only continues the test when both copies have performed the same event: so at all times both copies of P have performed the same trace.

We test this implementation against the specification below.

```
LHS_OutputDec_A(P) =
    STOP
    |~|
    ([] x:diff(Events,union(outputs(P),{clunk}))
        @ x -> LHS_OutputDec_A(P))
    []
    ([] x:outputs(P) @ x -> (|~| y:chan(x,P)
        @ y -> LHS_OutputDec_A(P)))
```

where

```
chan(ev,P) =
    inter(outputs(P),
          {| c | c <- GET_CHANNELS(P), member(ev,{|c|})|})
```

This will allow any trace that RHS_OutputDec_A can make, and only insists on some member of the same channel occurring after an output.

It is important to note that this certainly tests all traces of P since whenever one copy of P performs an event after trace t, it is certain that the other one can perform it, even though it may also be capable of refusing that event.

Thus, the refinement check below checks that, after every trace t of `P` after which an output can happen, the process cannot refuse the whole of the corresponding channel.

```
LHS_OutputDec_A [F= RHS_OutputDec_A(P)
```

This verification, however, does not check that the process can refuse all but one member of that channel. Hence, it does not check that the process is non-deterministic for a given output channel. For this reason, this verification accepts process that offer an external choice on the outputs of a same channel. Hence, a further check is needed to guarantee the non-deterministic choice on the outputs of the same channel.

Part B - Intra-channel Non-determinism. At this part of the verification, we need to guarantee that every single output of `P` can, if blocked, deadlock it on the same trace. To do this we need a Lazić construction on the left-hand side of the refinement check, along the lines of the process below.

```
LHS_OutputDec_B(P,c) =
    (FirstCopy(P)[|{clunk}|]SecondCopy(P))\{clunk}
    [|Events|]
    LHS_Test(inter({|c|},outputs(P)))
```

where

```
FirstCopy(P) = P [| AllButClunk |] DoubleClunker
SecondCopy(P) = P [| AllButClunk |] clunk -> DoubleClunker

DoubleClunker =
    [] x:AllButClunk @ x -> clunk -> clunk -> DoubleClunker

LHS_Test(S) =
    [] x:S @
        x -> (x -> LHS_Test(S) [>
                     ([] y:diff(S,{x}) @ y -> STOP)
                     []
                     ([] y:diff(Events,S) @ y -> STOP))
             [] ([] y:diff(Events,S) @ y -> y -> LHS_Test(S))
```

The process `LHS_OutputDec_B(P,c)` strictly alternates events of the two copies of the `P`, and as long as they have performed the same trace to date can, after a `c.x`, offer everything other than that event itself. So it ought, under what we want, to be able to refuse the whole of `{|c|}` after the first of each pair of `c.x` events.

We then check if that is refined by the same construction, replacing `RHS_Test` below.

```
RHS_Test(S) =
    [] x:S @
        x ->
            (([] y:S @ y -> if x==y then RHS_Test(S) else STOP)
                [> ([] y:diff(Events,S) @ y -> STOP))
            [] ([] y:diff(Events,S) @ y -> y -> RHS_Test(S))

RHS_OutputDec_B(P,c) =
    (FirstCopy(P)[|{clunk}|]SecondCopy(P))\{clunk}
    [|Events|]
    RHS_Test(inter({|c|}, outputs(P)))
```

Now, process `RHS_OutputDec_B(P,c)` behaves in the same way except that it can prevent the second process from performing the second of a pair of `c.x`'s. In both cases, the untimed time-out is used to create the possibility of repeating the event already input, without offering it in a stable way.

Combining Assertions. The final verification of Strong Output Decisiveness is then achieved in two parts. First, we verify the Part A, which is done for the whole process at once.

```
assert LHS_OutputDec_A [F= RHS_OutputDec_A(P)
```

If the assertion fails, the process is not Strong Output Decisive and the verification finishes. If, however, the process passes Part A, we need to check the Part B, which is done individually for every channel within the processes alphabet. For instance, supposing process αP = `{c1,c2}` we need the following assertions.

```
assert LHS_OutputDec_B(P,c1) [F= RHS_OutputDec_B(P,c1)
assert LHS_OutputDec_B(P,c2) [F= RHS_OutputDec_B(P,c2)
```

3.3.6 Further Side Conditions in CSP

Similar tricks were used to encode similar side conditions like checking if a channel is in the alphabet of a process. The assertions for decoupled channels (Prop. iv) ic and oc in P is encoded as a bi-directional refinement between the projection of P over both channels and the interleaving of the protocol implementation of P over each individual channel.

The finite output property (Prop. ii) has been characterised as an assertion that hiding all outputs of the protocol `P` does not introduce divergence:

```
  assert P \ allOutputs:[divergence free [FD]]
```

Furthermore, three theorems from [Ram11] were used in the definition of the characterisation tests. The first theorem (based on [Ram11]) states that a process P is I/O confluent (i) if, and only if, the process in which a one-place inwards-pointing buffer is placed on every individual event of $P \llbracket R \rrbracket$ (where R is a forgetful renaming that removes the data components of all channels but preserves their direction), is deterministic. Based on this theorem, our characterisation for processes P like our implementation protocols, that work in a single event `c` were defined as:

```
assert InBufferProt(P,c) :[deterministic [F]]
```

The second theorem states that protocols are strong compatible (Prop. iii) if one of them is a failures refinement of the dual protocol of the other. This allows us to characterise strong compatibility check as assertions on simple failures refinement.

Finally, a third theorem states that a buffering self-injection compatible (v) process can establish a communication between its channels via a one-place buffer without deadlock. This can be characterised as follows.

```
assert not PROJ(P,{i, o}) [| {| i, o |} |] BUFFIO(LR1, LR2)
           :[deadlock free [F]]
```

`LR1` and `LR2` provide the necessary renaming for communicating with `BUFFIO`.

Using these assertions, we were able to rigourously apply (and automatically verify) the systematic development approach to the case study presented in the next section.

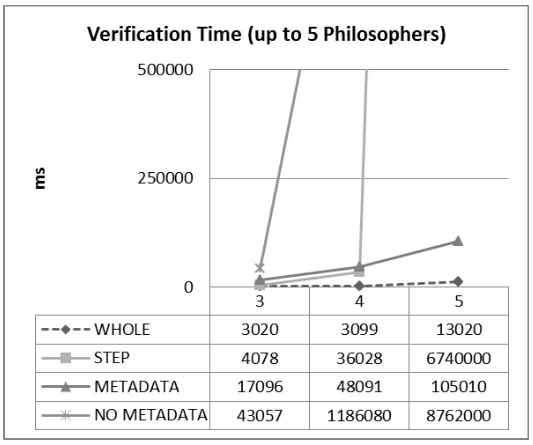
3.4 Experiments

The dining philosophers is a classical concurrency problem: n philosophers are seated at a round table with n forks and each fork is placed between each pair of philosophers. In order to eat, a philosopher must pick up the forks on either side. A philosopher who cannot pick up one or the other fork has to wait. However, since there is a limited number of forks, it is necessary to control the access to such resources. Otherwise, for instance, all philosophers might get hungry simultaneously and pick up one fork, then deadlock and starve to death.

The experiment consisted in verifying the CSP scripts of the dining philosophers using FDR, and collecting the overall verification time. The experiment was executed on a Intel Xeon CPU X3363, 2.83GHz, with 8Gb RAM, running Ubuntu 9.10 (Kernel 2.6.31-23 - 64 bits). The data were collected for both development approaches: standard deadlock check and a check of all side conditions required to apply the composition rules. Furthermore, the data for the standard deadlock check was collected for two different views: checking for deadlock after each composition (STEP), and checking for deadlock only at the final composition (WHOLE). Also, we consider the proposed rule-based strategy both with (METADATA) and without metadata (NO METADATA).

We performed a five-level analysis: each new level optimises the verification process by removing some of the side conditions based on theoretical results. Our experiment was executed in two phases. Our experiment was executed in two phases. The first phase considered a network of up to 5 philosophers (see Figure 5). It aimed to demonstrate the improvement in the verification time by using metadata. In this phase, the time without the use of metadata proved to be much higher than that with the use of metadata. This result demonstrated the infeasibility of the approach if metadata is not considered. Furthermore, the time for standard verification of a step-by-step view was also very high. Based on the results of the first phase, we focused on the most efficient verifications of both approaches in the second phase of the experiment, which considered a network of up to 7 philosophers. In Figure 6 we present the results of the original verification of the whole system and the systematic development with the use of metadata.

Concerning the effort for checking the conditions for the rule-based application we were able to get rid of the verification of some of the side conditions in both phases of the experiment. Simple conditions based on set theory may be verified by SAT solvers at a cost close to zero. For example, in Table 2, the first assertion refers to set containment and intersection checking that can be easily achieved using SAT solvers. Furthermore, since deadlock freedom is guaranteed by construction [ORS+12b], further application of composition rules to components that result from previous compositions do not need to check for their deadlock freedom. Also, further theorems guarantee deadlock freedom of protocol implementations of deadlock free processes [ORS+12b].



	3	4	5
WHOLE	3020	3099	13020
STEP	4078	36028	6740000
METADATA	17096	48091	105010
NO METADATA	43057	1186080	8762000

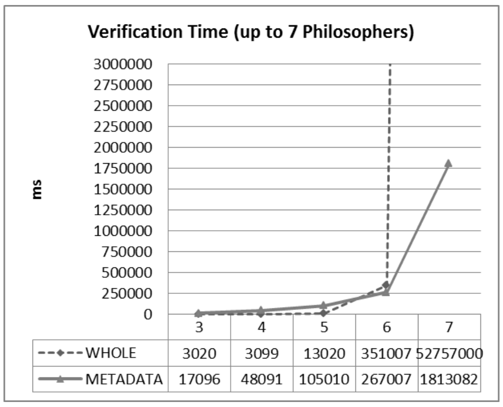
Figure 5: Experiment Results - Phase 1

In [Ros98], it is demonstrated that if a process has no hiding and no unguarded recursion, it is divergence free; these are syntactic restrictions that can be easily checked outside the scope of FDR. In [Ros98], it is also demonstrated that the checking for finite output property is irrelevant if we are using finite buffers. Finally, we also considered optimisations based on properties guaranteed by the extended rules that use process metadata calculated by construction (decoupled channels, protocol implementation and dual protocols) and optimisations based on properties guaranteed by previous theorems constructed for systems with replicated components like our case study (for example, different instances of philosophers and forks).

The results of the first phase are presented in Figure 5. At this phase of the experiment, the results of Levels 2 and 1 proved to be extremely high. For presentation purposes, we omitted this data in this figure.

The second phase of the experiment focused on the most optimised Levels 4 and 5. At this phase, the standard verification in a step-by-step view became much larger and, for presentation purposes, we omitted this data in the results presented in Figure 6.

The verification of side conditions at Level 1 and 2 proved to be extremely more expensive than both views of the standard verification. Overall, they presented a increase of over 50000% and 3000% if compared with the standard verification of the whole system and step-by-step, respectively. By taking into consideration the use of finite buffers at Level 3, the increase in the verification time was reduced to 40000% and 1500%, but was still unacceptable.



	3	4	5	6	7
WHOLE	3020	3099	13020	351007	52757000
METADATA	17096	48091	105010	267007	1813082

Figure 6: Experiment Results - Phase 2

The use of metadata, along with the optimisations presented above, proved to be the turning point in the experiment. Using metadata, the systematic approach presented a gain of 98.4% against the sum of the verification time of each individual composition in a 5 philosophers network (see Figure 5). It, however, presented a loss of 706.5% if compared with the verification of the overall system. The gains achieved with metadata proved to be higher as we increase the number of instantiations of the parameterised protocol implementations. In Figure 6, our approach offers a gain in verification time for networks with 6 philosophers or more. As a matter of fact, for networks with 7 philosophers, using metadata, our approach becomes one order of magnitude faster than the original approach. The systematic approach also provides a better understanding induced by an incremental and systematic system construction. Nevertheless, this gain was only possible based on the strong use of theoretical results (some based on the notion of metadata) that made it possible to take for granted side conditions that are needed for a valid composition rule application. This, however, considered the existence of an external proof that the parameterised processes are valid protocol instantiation using Theorem Provers like [IR08]. Indeed, this should involve the costs of interactive theorem proving, which is a well-known expensive activity. We, however, strongly believe that, based on syntactic restrictions, this activity may also be removed. Nevertheless, our approach presented a very small increase of 6.8% in the verification time. In our opinion this loss is relatively unimportant if we consider that the systematic development also offers a better understanding of the overall system construction.

The standard approach presented a much larger increase rate as we include more participants in the system. For instance, when increasing the number of philosophers from 6 to 7, the verification time in the standard approach (whole system) increases in 15000%, whilst this increase is of 580% in our approach.

Both approaches presented an exponential growth in Figure 6, in which the time for a network with 8 philosophers has been omitted for presentation purposes. In this network, the standard approach (whole system) took 460 hours to complete the check whilst our approach took 170 hours. Despite still presenting a significant improvement, the exponential growth indicated that scalability requires further investigation, as further discussed in the next section.

4 Lifting the Approach to Circus and CML

In this section, we lift the results presented in Section 3 to provide a similar systematic approach to build trustworthy CML systems. The main principle for lifting the approach from CSP to CML (via *Circus*) is to keep the main structure of the definitions and rules. The only change could be the references to CML processes and constructs rather than the corresponding CSP ones. Nevertheless, a thorough analysis indicated that some changes could be done to simplify the application of the approach. For example, we no longer require channels to communicates values `in` and `out` to indicate the direction of the communications. This considerably reduces the need for changes in the original CSP specification to which we want to apply the approach. Furthermore, in order to reuse these results by providing a theoretical link for processes and refinement, as we explain in Section 5, we restricted our scope to divergence-free processes. For this reason, the definition of channel projection, which uses hiding and potentially introduces divergence, required a new definition presented here.

For conciseness, in this section, we focus on the changes we applied to the original approach. The only change in the remaining definitions are the references to CML processes and constructs rather than the corresponding CSP ones. In Appendix H, we present a full Z type-checked formalisation of our compositional model.

The structure of this section is as follows: in Section 4.1 we present a simpler definition for component contracts; this simplification propagates to the definitions of renaming and I/O processes whose new definitions are presented in Sections 4.2 and 4.3, respectively; finally, new definitions for implementation protocols and channel projection that do not use hiding are presented in Section 4.4. These changes removed the possibility of divergence in the processes used in the approach. For this reason, when migrating from CSP into CML (via *Circus*) we are able to reuse the results from [Ram11] based on the links described in Section 5.

4.1 Component Contracts

Its original definition included a condition that forced a structure to the channels used in the process behaviour (either $c.*.in.*$ or $c.*.out.*$). Nevertheless, this definition already forces this behaviour to be an I/O process, which only uses I/O channels. Hence, every channel is either an input or output. This classification is not defined by the use of *in* or *out* in the communication value, but by the functions *inputs* and *outputs*. Therefore, we removed the third condition mentioned above.

New Definition 4.1 (Component contract) *A component contract Ctr comprises an observational behaviour $\mathcal{B}$, a set of communication channels $\mathcal{C}$, a set of interfaces $\mathcal{I}$, and a* ***total*** *function $\mathcal{R} : \mathcal{C} \to \mathcal{I}$ between channels and interfaces of the contract:*

$$Ctr : \langle \mathcal{B}, \mathcal{R}, \mathcal{I}, \mathcal{C} \rangle$$

such that $\mathcal{B}$ is an I/O process (a CML process that satisfy the conditions described in Definition 4.2).

4.2 Renaming Contracts

The renaming used in the composition rules was defined as follows:

$$R_{IO}^{a \to b} = \{a.out.x \mapsto b.in.x\}$$

As explained above in Section 4.1, there is no need to add *in* and *out* to the communication. Hence, we changed this definition to the following one.

$$P \llbracket R_{IO}^{a \to b} \rrbracket = P[\{a.x \mapsto b.x \mid a.x \in outputs(P)\}]$$

This corresponds to the original intention (i.e. replaces outputs of a by inputs of b) but uses the function *outputs* rather than the previously enforced channel structure.

There are also consequences to the definition of *inputs* and *outputs* of a renamed protocol.

$$inputs(P \llbracket R_{IO}^{a \to b} \rrbracket) = inputs(P)$$
$$outputs(P \llbracket R_{IO}^{a \to b} \rrbracket) = outputs(P)[a \mapsto b]$$

where $[a \mapsto b]$ replaces all references events on a to events on b in a given set of events.

$$S\,[a \mapsto b] = S \setminus \{a.x \mid a.x \in S\} \cup \{b.x \mid a.x \in S\}$$

4.3 I/O Processes

The definition of I/O Processes was also simplified by simply enforcing all channels used in the process to be I/O channels using a single condition.

New Definition 4.2 (I/O process) *We say P is a CML I/O process if:*

- *P only uses I/O Channels;*
- *P has infinite traces;*
- *P is divergent-free;*
- *P is input deterministic;*
- *P is strong output decisive.*

4.4 Implementation Protocols

The protocol implemented by a component (represented solely by a process at this point) is given by the abstraction of its behaviour projection over a specific channel. Moreover, the protocol has the same traces and failures as the projection, but it is divergent-free. We use the failures semantics here, since we ignore the possible divergences introduced by the restriction.

Definition 4.1 (Protocol implementation) *Let P be an I/O process, and ch a communication channel. The communication protocol $Prot_{IMP}(P, ch)$ implemented by P over ch is a protocol that satisfies the following property:*

$$Prot_{IMP}(P, ch) \equiv_{\mathrm{F}} P \upharpoonright ch$$

This definition might become unfeasible (there is simply no such implementation protocol that satisfies this property) in cases where $P \upharpoonright ch$ introduces divergences. This is due to the fact that in such cases $Prot_{IMP}(P, ch)$ might have failures that are not considered in $P \upharpoonright ch$ because they are in a non-stable (caused by the divergence) state. This invalidates the failures refinement from right to left. Moving to the failures-divergence refinement hits the same problem (in the other direction however) due to a simpler cause: $P \upharpoonright ch$ might introduce divergence.

A possible solution would be to consider different refinement scenarios in each direction as proposed below.

Definition 4.2 (Protocol implementation) *Let P be an I/O process, and ch a communication channel. The communication protocol $Prot_{IMP}(P, ch)$ implemented by P over ch is an I/O process that satisfies the following properties:*

- $Prot_{IMP}(P, ch) \sqsubseteq_{\mathrm{F}} P \upharpoonright ch$
- $P \upharpoonright ch \sqsubseteq_{\mathrm{FD}} Prot_{IMP}(P, ch)$

In either case, the definitions try to define a process that behaves just like P looking only at a given channel ch. However, by simply hiding it as in $P \upharpoonright ch$ we might introduce divergence and this is not a desirable property in an implementation protocol. Furthermore, in order to reuse the original CSP approach, we need to get rid of divergent processes since the theoretical link is provided only for divergence-free processes. In Appendix G we present an exercise on the new definition of channel projection that lead us to the divergence-free definition below. In this exercise, we investigate possible alternatives for redefining channel projection like using Roscoe's lazy abstraction as defined in [Ros98].

We propose the projection plays a role simply in the traces of a process. Hence, we would have that the projection on c of process P is a process that:

- Does not have any event other than c
- Has exactly the same traces as P on c; the behaviour on the other events are irrelevant.

For that, instead of calculating a given projection, the user of the strategy needs to propose a projection that satisfies these properties. This, however, might be automated by a syntactic function that removes the channel. Nevertheless, we also need to guarantee that the communication directions (input and output) are not changed and that the properties of strong output decisiveness and input determinism are maintained. Overall, these properties would be characterised as follows:

New Definition 4.3 (Projection) *Let P be an I/O Process, and C a set of communication channels. The projection of P over C (denoted by $P \upharpoonright C$) satisfies the following properties:*

1. $P \upharpoonright C$ *is an I/O Process*
2. $\forall\, c : C \bullet inputs(P \upharpoonright C, c) \subseteq inputs(P, c)$
3. $\forall\, c : C \bullet outputs(P \upharpoonright C, c) \subseteq outputs(P, c)$

4. $\alpha(P \restriction C) \subseteq \bigcup_{c:C} \{\!| \ c \ |\!\}$
5. $P \equiv_{\mathrm{T}} P \, [\![\, \Sigma \,]\!] \, ((P \restriction C) \, ||| \, RUN(NOT(C)))$

Properties 1 - 3 guarantees that the communication direction (input and output) are not changed and that the properties of strong output decisiveness and input determinism are maintained. This ensures that we are neither removing nor introducing non-determinism. Property four ensures that the projection process refers only to channels in C. Finally, together with the previous properties, property 5 guarantees that the process behaviour on the projected channels is not changed.

The changes to the definition of channel projection removed the possibility of divergence in the processes used in the approach because:

- There are no unguarded recursion;
- Hiding is not used;
- I/O Process are, by definition, divergence-free.

For this reason, when migrating from CSP into CML (via ***Circus***) we are able to reuse the results from [Ram11] based on the links described in Section 5.

5 Linking Theories

In this section, we provide a proof of the soundness of our technique for compositional reasoning about CML-based contracts. As already explained, this is achieved by lifting our results to CML. For pragmatic reasons, the strategy for providing the justification of this lift is twofold: first, in Section 5.3, we lift the strategy from CSP to *Circus*; finally, in Section 5.4, we lift the strategy from *Circus* to CML. The reason is that, due to the nature of the schedule of the COMPASS project, in which the development of this deliverable was done concomitantly with the development of the CML syntax and semantics. Hence, we adopted this strategy to first lift the whole systematic approach to a state-rich concurrent language, *Circus*, which has a structure and semantics similar to that of CML.

In both steps of the lifting strategy, two very important theoretical links are needed: processes and refinement. This is because the composition rules from [RSM10], their side conditions, and their semantical correctness are based on the definitions of CSP processes and CSP refinement ($\mathcal{T}$, $\mathcal{F}$ and $\mathcal{FD}$). Overall, a complete theoretical link for processes and refinement from CML into CSP is provided for a subset of CML, considering the following restrictions:

- untimed;
- feasible (no miracles);
- divergence free;
- no object-oriented constructs;
- no undefined expressions;
- limited use of predicative specifications;
- external choices are only among prefixed actions as defined in Appendix C;
- actions do not write to input variables.

This link allows us to reuse the results from [Ram11] in CML for such processes.

In Sections 5.1 and 5.2 we present an overview of the strategy to link CML processes and refinement to CSP processes and refinement, respectively. The details of the first part of the link, from *Circus* processes and refinement into CSP processes and refinement, is presented in Section 5.3 along with a discussion on the soundness of this mapping. Finally, in Section 5.4, we complete the link by providing the translation from CML to ***Circus***.

5.1 Linking Processes

In [RSM10], the behaviour of the basic components is defined in terms of CSP processes. The lifting of the results to CML (via Circus), requires two mappings: the first one maps CML processes to corresponding ***Circus*** processes; the second mapping is from a subset of ***Circus*** processes to corresponding CSP processes.

In the first mapping (ρ), we take a subset of CML processes without object-oriented constructs and without undefined expressions and return the semantically corresponding ***Circus*** process. In the second mapping, we take processes from a state-rich setting, ***Circus***, to a stateless one, CSP. For this reason, our strategy for mapping ***Circus*** processes into CSP processes depicted in Figure 7 is twofold: transforming stateful ***Circus*** processes into stateless ***Circus*** processes (Ω), and mapping a subset of stateless divergence-free ***Circus*** processes with a limited use of predicative specifications into corresponding CSP processes (Υ).

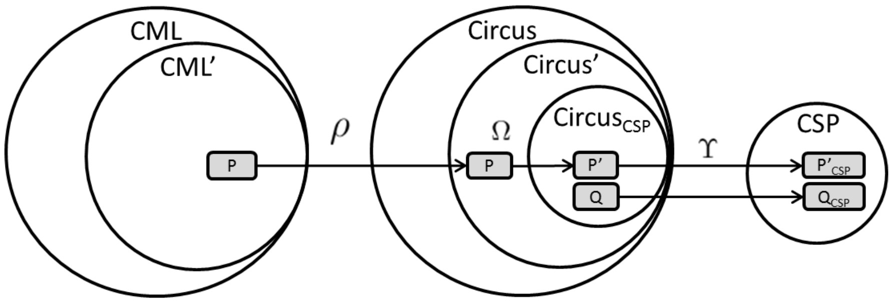


Figure 7: Mapping CML into CSP

In our mapping strategy from ***Circus*** to CSP, we first consider **stateful *Circus*** processes, that is, processes with encapsulated states and local variables. Instead of mapping such processes directly into CSP, we first transform them, using a function Ω, into stateless processes using the memory model suggested in [NSM12], in which state components and local variables are detached from the processes and moved to memory cells that store their values. The soundness of this transformation is established using the ***Circus*** refinement calculus presented in [Oli06].

The function Ω takes a subset of **stateful feasible *Circus*** processes, in which:

1. External choices are only among prefixed actions as defined in Appendix C;
2. Actions do not write to input variables;
3. Actions do not present a miraculous behaviour like, for example, infeasible specification statements.

All these restrictions on Ω could be relaxed in order to broaden the application of this function. For instance, Ω could take infeasible actions like *Miracle* and return the action itself. Nevertheless, for simplification purposes, we restricted the domain of Ω by removing infeasible actions from its domain (*Circus*'). As a consequence, we are able to define the function that transforms *Circus* processes into CSP processes, Υ, as a total function on $Circus_{CSP}$, the range of Ω. Hence, $Circus_{CSP}$ constitutes the set of *Circus* processes that can be directly translated into their corresponding CSP processes.

Next, we provide a mapping Υ for a subset of **stateless *Circus*** processes into corresponding CSP processes. This subset contains all stateless processes whose main actions are defined only in terms of *Circus* behavioural actions, that is, those actions that are directly available in CSP. This includes *Skip*, *Stop*, prefixing, external and internal choice, guarded action, sequential composition, parallelism, interleaving, hiding, recursion and the iterated operators. This mapping guarantees that the *Circus* processes used to define a component's behaviour at the *Circus* level have a corresponding CSP behaviour that defines the corresponding component's behaviour at the CSP level. The soundness of this mapping Υ is established for the traces and the failures models. The establishment of correctness of the mapping in these models is enough for our purposes since we consider only divergent free processes (See Definition 3.1 in Section 3). For every *Circus* action A that is mapped into a CSP process P we prove that the traces of A (in the UTP) are the same as those of P (in CSP). We do the same for the failures model.

The details of the mapping from *Circus* to CSP are discussed in Section 5.3.

5.2 Linking Refinement

The application of the composition rules is only allowed under certain conditions. Some of these conditions are refinement based ($\mathcal{T}$, $\mathcal{F}$ and $\mathcal{FD}$). Hence, the lifting of the systematic composition approach to CML, $\mathcal{BRIC}$, requires a relation between CML refinement and CSP refinement (again via *Circus*).

The second part of this mapping, from *Circus* to CSP, is based on the work of Cavalcanti and Gaudel briefly discussed in Section 5.3, which provides a connection between the *Circus* and CSP theories within the UTP. Nevertheless, since the original systematic approach is underpinned by the original CSP semantics, again, a mapping from the subset of *Circus* actions (which can be expressed in CSP) into the corresponding CSP processes is required.

In Section 5.4, we present the strategy (and discuss its correctness) for mapping CML processes into *Circus* processes. Next, in Section 5.3, we present the strategy for mapping *Circus* processes into CSP processes. Section 5.3.2 briefly introduces the work by Cavalcanti et al which provides a link between *Circus* and CSP theories within the Unifying Theories of Programming (UTP) [HJ98] and discusses the correctness of our approach.

5.3 From *Circus* to CSP

In this section we present the mapping from *Circus* processes and refinement into CSP processes and refinement. First, in Section 5.3.1, we present the strategy to map stateful *Circus* processes into CSP processes. Finally, we present the proof of correctness of this mapping in Section 5.3.2.

5.3.1 Mapping Circus into CSP

In Figure 7, we presented the overall idea of the strategy for mapping *Circus* processes into CSP processes, which applies to a subset of feasible *Circus* processes that can be mapped into CSP. First, we transform processes with encapsulated states and local variables into stateless processes using the memory model suggested in [NSM12]. In his work, Nogueira detaches state components and local variables from the processes and moves them to a separate memory process that stores their values. The result is an equivalent process in which stateless processes communicate with a *Memory* process that encapsulates the original process components. Next, we provide a mapping for **stateless *Circus*** processes into corresponding CSP processes.

For simplicity, we consider:

1. External choices are only among prefixed actions as defined in Appendix C. This removes the possibility of actions like $(x := e;\ A_1) \Box A_2$ in which, as described in [Oli06], the assignment does not solve the choice. The removal of the state components from processes with such actions requires the use of a protocol that adds an overhead which we avoid with this restriction;

2. Input variables are not written. By way of illustration, consider the action $c?x \rightarrow A(x)$. This restriction forbids that the value of x is updated in $A(x)$.

The restrictions have impact on the specification style, but do not impose any relevant limitation in terms of expressiveness. Divergence-free processes that satisfy these conditions are within the domain of Ω and suitable for transformation into CSP.

The basic structure of stateless *Circus* processes in *Circus*' is:

$$
\begin{array}{l}
\textbf{process}\ Q \mathrel{\widehat=} \\
\quad \textbf{begin} \\
\qquad \bullet\ A_{CSP} \\
\quad \textbf{end}
\end{array}
$$

As they do not have state components and local variables, they are also members of *Circus*$_{CSP}$: their translation into corresponding CSP processes is defined by the fairly direct mapping function Υ whose details and correctness are omitted here and presented in Appendixes B and J. In Figure 7, the process Q falls into this category.

The second class of *Circus* processes (i.e. P in Figure 7) corresponds to those processes that have state components and local variables in the main action. In Figure 7, the process P falls into this category.

$$
\begin{array}{l}
\textbf{process}\ P \mathrel{\widehat=} \\
\quad \textbf{begin} \\
\qquad \textbf{state}\ S \mathrel{\widehat=}\ [v_0 : T_x;\ \ldots\ v_n : T_z \mid inv(v_0, \ldots, v_n)] \\
\qquad \bullet\ \textbf{var}\ l_0 : U_0;\ \ldots;\ l_m : U_m \bullet A(v_0, \ldots, v_n, l_0, \ldots, l_m) \\
\quad \textbf{end}
\end{array}
$$

As expected, these processes are also supposed to satisfy the restrictions discussed in the beginning of this section.

The strategy is to transform these processes using a translation function Ω. This function moves the state components and local variables from the processes to a separate *Memory* action that encapsulates the state components and local variables. In Figure 7, the process P is transformed into a stateless process P', which can then be transformed using the mapping function Υ.

Each of the state components and local variables have a corresponding member in the set of names $NAME$.

$$\textbf{nameset}\ NAME == \{v_0, \ldots, v_n, l_0, \ldots, l_n\}$$

We consider that a pre-processing of the ***Circus*** processes determines this set of names.

A set of functions, $BINDING$, represents all possible mappings from names to values.

$$BINDING \mathrel{\widehat=} NAME \rightarrow \mathbb{U}$$

The process that describes the behavioural aspects of the original process interacts with the memory either by requesting a variable value using a channel *get* or by setting a variable new value using channel *set*. Furthermore, as described below, the memory has a recursive behaviour. When the behaviour of the original process terminates, the memory is also requested to terminated via the channel *terminate*. These channels are considered as the memory interface, characterised by the set MEM_I.

$$\textbf{channel}\ get, set : NAME \times \mathbb{U}$$
$$\textbf{channel}\ terminate$$

$$MEM_I \mathrel{\widehat=} \{\!|\ set, get, terminate\ |\!\}$$

The resulting process is a stateless version of the original process P. Its main action is the parallel composition of a memory action with a stateless version of the original main action A. The range of the translation function Ω is within the subset ***Circus***$_{CSP}$ of ***Circus*** processes that can be directly translated into their corresponding CSP processes. Hence, the resulting parallel composition (i.e. P' in Figure 7) presented below has a corresponding pure CSP behaviour.

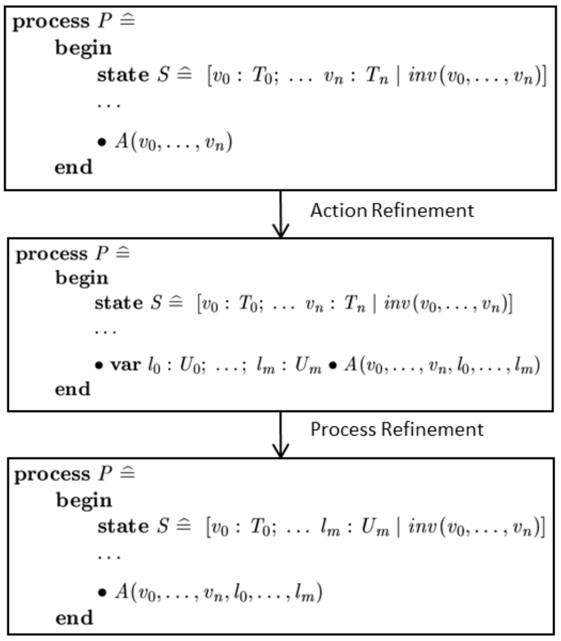
$$
\begin{array}{l}
\Omega\,(P) \mathrel{\widehat=} \\
\quad \mathbf{process}\ P' \mathrel{\widehat=} \\
\quad\quad \mathbf{begin} \\
\quad\quad\quad \mathit{Memory} \mathrel{\widehat=} \\
\quad\quad\quad\quad \mathbf{vres}\ b : \mathit{BINDING} \bullet \\
\quad\quad\quad\quad\quad (\Box\, n : \operatorname{dom} b \bullet \mathit{get}.n!b(n) \to \mathit{Memory}(b)) \\
\quad\quad\quad\quad\quad \Box \left(\begin{array}{l} \Box\, n : \operatorname{dom} b \bullet \\ \quad \mathit{set}.n?nv : (nv \in \delta(n)) \to \\ \quad \mathit{Memory}(b \oplus \{n \mapsto nv\}) \end{array} \right) \\
\quad\quad\quad\quad\quad \Box\ \mathit{terminate} \to \mathit{Skip} \\
\quad\quad\quad \bullet\ \mathbf{var}\ b : \left\{ \begin{array}{l} x : \mathit{BINDING} \mid x(v_0) \in T_0 \land \ldots \\ \qquad \land\ \mathit{inv}(x(v_0), \ldots, x(v_n)) \end{array} \right\} \bullet \\
\quad\quad\quad\quad \left(\begin{array}{l} \left(\begin{array}{l} \Omega_A(A); \\ \mathit{terminate} \to \mathit{Skip} \end{array} \right) \\ [\![\emptyset \mid \mathit{MEM}_I \mid \{b\}]\!] \\ \mathit{Memory}(b) \end{array} \right) \setminus \mathit{MEM}_I \\
\quad\quad \mathbf{end}
\end{array}
$$

The approach of a centralised memory considerably simplifies the proof of correctness discussed in Section 5.3.2. The simplification in the proof effort is due to the fact that a centralised memory uses no replicated operators, which currently are not well covered by the existing *Circus* refinement laws. Furthermore, by not using replicated operators we also avoid the need for induction in the proofs. This induction is needed because using a distributed memory, the proofs are achieved by induction on the number of memory cells used in the parallel composition of the *Memory* definition.

Using the same refinement strategy as that presented in [CSW05a] we may demonstrate that a distributed memory is a refinement of the centralised memory. In the distributed memory, independent memory cells are responsible for storing the values of each state component variable. The overall memory is the parallel composition of all memory cells synchronising on the termination event only. The monotonicity of *Circus* refinement allows us to simply replace the centralised memory by its distributed version if it turns out to be more convenient.

Rewriting *Circus* Processes (Definition of Ω) The transformation Ω can be formalised using the *Circus* refinement calculus presented in [Oli06] in which two refinement iterations of the *Circus* refinement strategy are used.

The first iteration, presented in Figure 8, aims at adapting the process to the translation restrictions discussed on page 79 like copy-rule application, renaming variables and channels, and schema normalisation. Furthermore, this iteration also aims at promoting all local variables to state components using the strategy presented in [CCO11]. First, an action refinement adapts the process to the translation restrictions: renames all local variables to avoid name clashes. Finally, it moves the variable declarations to the outermost scope in the process main action. For that, we may use refinement laws like Law 4 (var-exp-seq). We are then able to make a process refinement using Laws 26 and 27 to promote the local variables to state components. This facilitates the data refinement of the second iteration in which state components are replaced by a single mapping function as we describe below.


process $P \widehat{=}$
    begin
        state $S \widehat{=} [v_0 : T_0; \ldots\ v_n : T_n \mid inv(v_0, \ldots, v_n)]$
        $\ldots$
        $\bullet\ A(v_0, \ldots, v_n)$
    end

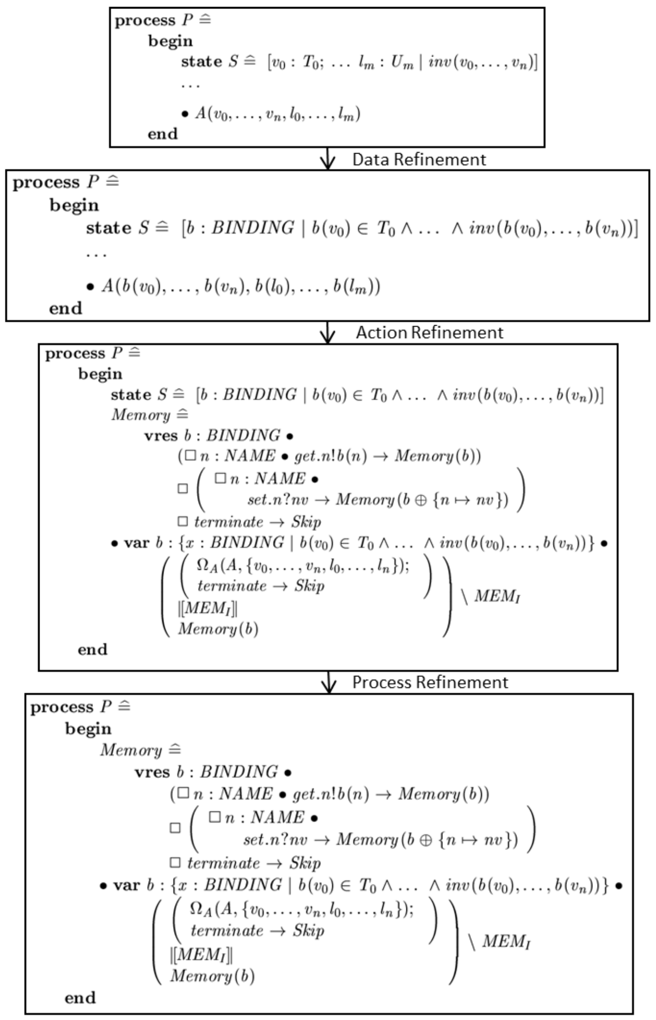
process $P \widehat{=}$
    begin
        state $S \widehat{=} [v_0 : T_0; \ldots\ v_n : T_n \mid inv(v_0, \ldots, v_n)]$
        $\ldots$
        $\bullet$ **var** $l_0 : U_0; \ldots; l_m : U_m \bullet A(v_0, \ldots, v_n, l_0, \ldots, l_m)$
    end

process $P \widehat{=}$
    begin
        state $S \widehat{=} [v_0 : T_0; \ldots\ l_m : U_m \mid inv(v_0, \ldots, v_n)]$
        $\ldots$
        $\bullet\ A(v_0, \ldots, v_n, l_0, \ldots, l_m)$
    end

Figure 8: First Iteration of Refinement Strategy

The final iteration, presented in Figure 9, aims at transforming the stateful process into a stateless process. For that, we first make a data refinement to transform the state from a state with multiple components into a state with a single binding component that maps the original state component names into their values. Next, an action refinement transforms the centralised stateful main action into a stateless main action in which the transformed main action

$\Omega_A(A)$ interacts with a memory that stores the values of the state components from the original process. Finally, since the state components are no longer referenced in the resulting main action, we apply a process refinement that completely removes the process state.



```
process P ≙
    begin
        state S ≙ [v_0 : T_0; ... l_m : U_m | inv(v_0, ..., v_n)]
        ...
        • A(v_0, ..., v_n, l_0, ..., l_m)
    end
```

```
process P ≙
    begin
        state S ≙ [b : BINDING | b(v_0) ∈ T_0 ∧ ... ∧ inv(b(v_0), ..., b(v_n))]
        ...
        • A(b(v_0), ..., b(v_n), b(l_0), ..., b(l_m))
    end
```

```
process P ≙
    begin
        state S ≙ [b : BINDING | b(v_0) ∈ T_0 ∧ ... ∧ inv(b(v_0), ..., b(v_n))]
        Memory ≙
            vres b : BINDING •
                (□ n : NAME • get.n!b(n) → Memory(b))
                □ ( □ n : NAME •
                        set.n?nv → Memory(b ⊕ {n ↦ nv}) )
                □ terminate → Skip
        • var b : {x : BINDING | b(v_0) ∈ T_0 ∧ ... ∧ inv(b(v_0), ..., b(v_n))} •
            ( ( Ω_A(A, {v_0, ..., v_n, l_0, ..., l_n});
                terminate → Skip )
              ⟦MEM_I⟧
              Memory(b) ) \ MEM_I
    end
```

```
process P ≙
    begin
        Memory ≙
            vres b : BINDING •
                (□ n : NAME • get.n!b(n) → Memory(b))
                □ ( □ n : NAME •
                        set.n?nv → Memory(b ⊕ {n ↦ nv}) )
                □ terminate → Skip
        • var b : {x : BINDING | b(v_0) ∈ T_0 ∧ ... ∧ inv(b(v_0), ..., b(v_n))} •
            ( ( Ω_A(A, {v_0, ..., v_n, l_0, ..., l_n});
                terminate → Skip )
              ⟦MEM_I⟧
              Memory(b) ) \ MEM_I
    end
```

Figure 9: Second Iteration of Refinement Strategy

The function Ω_A rewrites the ***Circus*** actions into their corresponding stateless ***Circus*** actions that considers the interaction with the memory process. In what follows, we present its definition by induction on the syntax of ***Circus*** actions. The correctness of the rewriting function Ω is proved using the ***Circus*** refinement calculus as we discuss in Section 5.3.2.

Rewriting *Circus* Actions (Definition of Ω_A) The main principle of Ω_A is to change only actions that access state components and local variables (memory components). Its definition uses an auxiliary function Ω'_A that is very similar to Ω_A, but does not retrieve any value (get) and replaces references to x by its local copy vx, except when used as the identifier in memory access ($i.e.\ set.x!e(x)$ becomes $set.x!e(vx)$). For sequential composition, the difference is more substantial as we discuss later in this section.

The main principle behind the definition of the function Ω_A for ***Circus*** actions is to change only those actions that access state components and local variables either by reading or writing on them. Actions that do not present such behaviour remain unchanged. For instance, the three basic functions *Skip*, *Stop* and *Chaos* remain unchanged.

$$\Omega_A(\mathit{Skip}) \mathrel{\widehat=} \mathit{Skip}$$

$$\Omega_A(\mathit{Stop}) \mathrel{\widehat=} \mathit{Stop}$$

$$\Omega_A(\mathit{Chaos}) \mathrel{\widehat=} \mathit{Chaos}$$

The transformation of prefixing actions differs according to the communication. Simple prefixing does not refer to state components and local variables: its rewriting leaves the communication unchanged and propagates the transformation to the action that follows the communication.

$$\Omega_A(c \to A) \mathrel{\widehat=} c \to \Omega_A(A)$$

Nevertheless, output communications ($c!e$) and synchronisation ($c.e$) might refer to state components ($v_0, \ldots, v_n$) and local variables ($l_0, \ldots, l_m$) in the expression used to define the communicated values. For this reason, before the original communication, the rewritten action needs to receive their values from the memory before the actual communication, which uses these values to define the communicated values.

$$\begin{array}{l} \Omega_A(c.e(v_0, \ldots, v_n, l_0, \ldots, l_m) \to A) \mathrel{\widehat=} \\ \quad get.v_0?vv_0 \to \cdots \to get.v_n?vv_n \to \\ \quad get.l_0?vl_0 \to \cdots \to get.l_n?vl_n \to \\ \quad c.e(vv_0, \ldots, vv_n, vl_0, \ldots, vl_m) \to \Omega'_A(A) \end{array}$$

$$\begin{array}{l} \Omega_A(c!e(v_0, \ldots, v_n, l_0, \ldots, l_m) \to A) \mathrel{\widehat=} \\ \quad \Omega_A(c.e(v_0, \ldots, v_n, l_0, \ldots, l_m) \to A) \end{array}$$

This approach is used to rewrite all ***Circus*** actions that need a read access to state components and local variables. For instance, guarded actions use the values received from the memory in the predicate that guards the referred action.

$$\begin{array}{l}\Omega_A(g(v_0, \ldots, v_n, l_0, \ldots, l_m) \mathbin{\&} A) \mathrel{\widehat{=}} \\ \qquad get.v_0?vv_0 \rightarrow \cdots \rightarrow get.v_n?vv_n \rightarrow \\ \qquad get.l_0?vl_0 \rightarrow \cdots \rightarrow get.l_n?vl_n \rightarrow \\ \qquad g(vv_0, \ldots, vv_n, vl_0, \ldots, vl_m) \mathbin{\&} \Omega'_A(A)\end{array}$$

On the other hand, the input prefixing $c?x : P \rightarrow A(x)$ defines the current value of the input variable. We, however, restrict the access mode to input variables like x in the action $A(x)$ that follows the input prefixing: x cannot be written by $A(x)$ $(x \notin wrtV(A))$. In the presence of such use of input variable, the ***Circus*** refinement calculus might be used to introduce an auxiliary variable and use it accordingly as a means to remove the direct writing to x. The input prefixing may be associated with a condition P that determines the values that may be communicated by restricting them to only those that satisfy P. For this reason, input prefixing also needs read access to the state components and local variables. Hence, the rewritten action also receives these values before the actual input communication.

$$\begin{array}{l}\Omega_A(c?x : P(x, v_0, \ldots, v_n, l_0, \ldots, l_m) \rightarrow A) \mathrel{\widehat{=}} \\ \qquad get.v_0?vv_0 \rightarrow \cdots \rightarrow get.v_n?vv_n \rightarrow \\ \qquad get.l_0?vl_0 \rightarrow \cdots \rightarrow get.l_n?vl_n \rightarrow \\ \qquad c?x : P(x, vv_0, \ldots, vv_n, vl_0, \ldots, vl_m) \rightarrow \Omega'_A(A) \\ \mathbf{provided}\ x \notin wrtV(A)\end{array}$$

It is important to notice that, because input variables are not part of the memory, there is no need to write x to it.

The rewriting of sequential composition and internal choice simply propagates to the composing actions.

$$\Omega_A(A_1;\ A_2) \mathrel{\widehat{=}}\ \Omega_A(A_1);\ \Omega_A(A_2)$$

$$\Omega_A(A_1 \sqcap A_2) \mathrel{\widehat{=}}\ \Omega_A(A_1) \sqcap \Omega_A(A_2)$$

It is important to notice that the memory model used in our approach allowed the distribution of Ω_A over sequential composition.

The rewriting of external choice requires the actions involved to be prefixed. The noise of the *mget* events is avoided by performing them before the choice.

$$\begin{array}{l}
\Omega_A(A_1 \mathbin{\Box} A_2) \mathrel{\widehat=} \\
\quad get.v_0?vv_0 \to \cdots \to get.v_n?vv_n \to \\
\quad get.l_0?vl_0 \to \cdots \to get.l_n?vl_n \to \\
\quad (\Omega'_A(A_1) \mathbin{\Box} \Omega'_A(A_2)) \\
\textbf{provided } A_1 \text{ and } A_2 \text{ are prefixed actions as defined in Appendix C}
\end{array}$$

In parallel composition (and interleaving), ***Circus*** avoids conflicts in the access to the variables by declaring two disjoint sets of variables. In $A_1 \llbracket ns_1 \mid cs \mid ns_2 \rrbracket A_2$, both A_1 and A_2 have access to the initial values of all variables, but A_1 may modify only the variables in ns_1, and A_2, the variables in ns_2. Our rewriting function uses two copies of the main memory, one for each parallel branch. A merge writes the final values to the main memory according to the state partition.

$$\begin{array}{l}
\Omega_A(A_1 \llbracket ns_1 \mid cs \mid ns_2 \rrbracket A_2) \mathrel{\widehat=} \\
\quad get.v_0?vv_0 \to \cdots \to get.l_0?vl_0 \to \cdots \to \\
\quad \left(\begin{array}{l}
\left(\begin{array}{l}
\left(\begin{array}{l}
(\Omega'_A(A_1);\ terminate \to Skip) \\
\llbracket \emptyset \mid MEM_I \mid \emptyset \rrbracket \\
MemoryMerge(\{v_0 \mapsto vv_0, \ldots\}, LEFT)
\end{array} \right) \setminus MEM_I \\
\llbracket \emptyset \mid cs \mid \emptyset \rrbracket \\
\left(\begin{array}{l}
(\Omega'_A(A_2);\ terminate \to Skip) \\
\llbracket \emptyset \mid MEM_I \mid \emptyset \rrbracket \\
MemoryMerge(\{v_0 \mapsto vv_0, \ldots\}, RIGHT)
\end{array} \right) \setminus MEM_I
\end{array} \right) \\
\llbracket \emptyset \mid MRG_I \mid \emptyset \rrbracket \\
Merge
\end{array} \right) \\
\quad \setminus \{\!| mleft, mright |\!\} \\
\textbf{where } Merge \mathrel{\widehat=} \begin{array}[t]{l}
(mleft?l \to (\mathbin{;}\ n : ns_1 \bullet set.n!l(n) \to Skip)) \\
\interleave (mright?r \to (\mathbin{;}\ n : ns_2 \bullet set.n!r(n) \to Skip))
\end{array}
\end{array}$$

The local memory *MemoryMerge* behaves like *Memory*, but writes its final bindings either to *mleft* or to *mright* after termination, based on the side given as argument.

$$
\begin{array}{l}
\mathit{MemoryMerge} \mathrel{\widehat=} \\
\quad \mathbf{vres}\ b : \mathit{BINDING};\ s : \mathit{SIDE} \bullet \\
\qquad (\Box\, n : \mathrm{dom}\, b \bullet \mathit{get}.n!b(n) \rightarrow \mathit{MemoryMerge}(b, s)) \\
\qquad \Box \left(\begin{array}{l} \Box\, n : \mathrm{dom}\, b \bullet \\ \quad \mathit{set}.n?nv : (nv \in \delta(n)) \rightarrow \\ \quad \mathit{MemoryMerge}(b \oplus \{n \mapsto nv\}, s) \end{array} \right) \\
\qquad \Box\ \mathit{terminate} \rightarrow \\
\qquad\quad (s = \mathit{LEFT})\ \&\ \mathit{mleft}!b \rightarrow \mathit{Skip} \\
\qquad\quad \Box\ (s = \mathit{RIGHT})\ \&\ \mathit{mright}!b \rightarrow \mathit{Skip}
\end{array}
$$

Before termination, each parallel branch communicates with *Merge* using *mleft* and *mright*, which are hidden from the environment.

$$\mathit{MRG}_I \mathrel{\widehat=} \{\!|\ \mathit{mleft}, \mathit{mright}\ |\!\}$$

The *Merge* receives the bindings and writes to the main memory based on the partitions.

Next, rewriting simply propagates through hiding, instantiation of unnamed parameterised actions and recursion.

$$\Omega_A(A \setminus cs) \mathrel{\widehat=} \Omega_A(A) \setminus cs$$

$$\Omega_A((x : T \bullet A(x))(e)) \mathrel{\widehat=} \Omega_A(A[e/x])$$

$$\Omega_A(\mu\, X \bullet A(X)) \mathrel{\widehat=} \mu\, X \bullet \Omega_A(A(X))$$

The transformation of iterated actions simply rewrites the expanded versions of the actions.

$$\Omega_A(\fatsemi\ x : \langle v_1, \ldots, v_n\rangle \bullet A(x)) \mathrel{\widehat=} \Omega_A(A(v_1);\ \ldots;\ A(v_n))$$
$$\Omega_A(\Box\, x : T \bullet A(x)) \mathrel{\widehat=} \Omega_A(A(v_1) \Box \cdots \Box A(v_n))$$
$$\Omega_A(\sqcap\, x : T \bullet A(x)) \mathrel{\widehat=} \Omega_A(A(v_1) \sqcap \cdots \sqcap A(v_n))$$
$$\Omega_A(\lbrack\!\lbrack\, cs \,\rbrack\!\rbrack\ x : \{v_1, \ldots, v_n\} \bullet \lbrack\!\lbrack\, ns(x) \,\rbrack\!\rbrack\ A(x)) \mathrel{\widehat=}$$
$$\Omega_A\left(\begin{array}{l} A(v_1)) \\ \lbrack\!\lbrack ns(v_1) \mid cs \mid \bigcup\{x : \{v_2, \ldots, v_n\} \bullet ns(x)\}\rbrack\!\rbrack \\ \left(\ldots\left(\begin{array}{l}\Omega_A(A(v_{n-1})) \\ \lbrack\!\lbrack ns(v_{n-1}) \mid cs \mid ns(v_n)\rbrack\!\rbrack \\ A(v_n)\end{array}\right)\right)\end{array}\right)$$
$$\Omega_A(|\!|\!|\ x : \{v_1, \ldots, v_n\} \bullet |\!|\!\lbrack\, ns(x)\,\rbrack\!|\!|\ A(x)) \mathrel{\widehat=}$$
$$\Omega_A\left(\begin{array}{l} A(v_1) \\ |\!|\!\lbrack ns(v_1) \mid \bigcup\{x : \{v_2, \ldots, v_n\} \bullet ns(x)\}\rbrack\!|\!| \\ \left(\ldots\left(\begin{array}{l}A(v_{n-1}) \\ |\!|\!\lbrack ns(v_{n-1}) \mid ns(v_n)\rbrack\!|\!| \\ A(v_n)\end{array}\right)\right)\end{array}\right)$$

This concludes the definition of the rewriting function Ω_A for ***Circus*** CSP-based actions. We now turn to the ***Circus*** commands.

In the semantic model for ***Circus*** processes presented in [Oli06], nothing is explicitly stated about the invariant. We assume specifications that initially contain no commands and, therefore, change the state using only Z operations, which explicitly include the state invariant and guarantee that it is maintained. For this reason, the semantics ignores any existing state invariants, since they are considered in the refinement process, just as in Z. Hence, the translation of the commands that follows also ignores state invariants.

In general, the commands that might potentially change the state need to be completely rewritten as its potential change need to be written to the newly introduced memory. This is the case, for instance, for an assignment, which is rewritten as a sequence of gets and the respective sets.

$$\Omega_A\left(x_0, \ldots, x_n := e_0\left(\begin{array}{c}v_0, \ldots, v_n, \\ l_0, \ldots, l_m\end{array}\right), \ldots, e_n\left(\begin{array}{c}v_0, \ldots, v_n, \\ l_0, \ldots, l_m\end{array}\right)\right) \mathrel{\widehat=}$$
$$\begin{array}{l} get.v_0?vv_0 \rightarrow \cdots \rightarrow get.v_n?vv_n \rightarrow \\ get.l_0?vl_0 \rightarrow \cdots \rightarrow get.l_n?vl_n \rightarrow \\ set.x_0!e_0(vv_0, \ldots, vv_n, vl_0, \ldots, vl_m) \rightarrow \\ \cdots \rightarrow \\ set.x_n!e_n(vv_0, \ldots, vv_n, vl_0, \ldots, vl_m) \rightarrow Skip \end{array}$$

The definition for alternation is relatively simple. As its conditions might refer to state components $(v_0, \ldots, v_n)$ and local variables $(l_0, \ldots, l_m)$ the rewritten action needs to receive their values from the memory before the rewritten alternation which uses these values to define the conditions.

$$
\begin{array}{l}
\Omega_A \left(\begin{array}{l}
\textbf{if } g_0(v_0, \ldots, v_n, l_0, \ldots, l_m) \rightarrow \ A_0 \\
\quad [\!] \ \cdots \\
\quad [\!] \ g_n(v_0, \ldots, v_n, l_0, \ldots, l_m) \rightarrow \ A_n \\
\textbf{fi}
\end{array} \right) \mathrel{\widehat=} \\
\quad get.v_0?vv_0 \rightarrow \cdots \rightarrow get.v_n?vv_n \rightarrow \\
\quad get.l_0?vl_0 \rightarrow \cdots \rightarrow get.l_n?vl_n \rightarrow \\
\quad \textbf{if } g_0(vv_0, \ldots, vv_n, vl_0, \ldots, vl_m) \rightarrow \ \Omega'_A(A_0) \\
\qquad [\!] \ \cdots \\
\qquad [\!] \ g_n(vv_0, \ldots, vv_n, vl_0, \ldots, vl_m) \rightarrow \ \Omega'_A(A_n) \\
\quad \textbf{fi}
\end{array}
$$

Specification statements might define a miraculous behaviour. We, however, remove this from the domain of the Ω_A function as we are not able to describe such behaviours in CSP. For that, however, we define proof obligations in the definition of Ω_A, which need to be discharged to validate the function application. The stateless action resulting from rewriting a specification statement, after getting the values of the variables in scope, might diverge if the precondition is not satisfied. Otherwise, the action non-deterministically chooses values for the variables in scope such that: (1) the postcondition is satisfied; (2) the values of the variables that are not in the frame w $(\overline{w})$ are not changed; and the invariant is respected. These values are finally written to the memory. In order to guarantee feasibility, the rewriting may only take place if such values exist.

$$
\begin{array}{l}
\Omega_A \left(w : \left[\begin{array}{l} pre(v_0, \ldots, v_n, l_0, \ldots, l_m), \\ post \left(\begin{array}{l} v_0, \ldots, v_n, l_0, \ldots, l_m, \\ v'_0, \ldots, v'_n, l'_0, \ldots, l'_m \end{array} \right) \end{array} \right] \right) \mathrel{\widehat=} \\
\quad get.v_0?vv_0 \rightarrow \cdots \rightarrow get.v_n?vv_n \rightarrow \\
\quad get.l_0?vl_0 \rightarrow \cdots \rightarrow get.l_n?vl_n \rightarrow \\
\quad \left(\begin{array}{l}
\neg\, pre(vv_0, \ldots, vv_n, vl_0, \ldots, vl_m)\ \&\ Chaos \\
\Box\ pre(vv_0, \ldots, vv_n, vl_0, \ldots, vl_m)\ \& \\
\quad \left(\begin{array}{l}
\sqcap\, vv : \left\{ \begin{array}{l} x_0 : \Gamma(v_0);\ \ldots;\ x_n : \Gamma(v_n); \\ \quad x_{n+1} : \Gamma(l_0);\ \ldots\ x_m : \Gamma(l_m) \\ \mid post \left(\begin{array}{l} vv_0, \ldots, vv_n, vl_0, \ldots, vl_m, \\ x_0, \ldots, x_n, x_{n+1}, \ldots, x_m \end{array} \right) \\ \wedge\ \overline{w}' = \overline{w} \\ \bullet\ (x_0, \ldots, x_n, x_{n+1}, \ldots, x_m) \end{array} \right\} \bullet \\
\qquad set.v_0!(vv.0) \rightarrow \cdots \rightarrow \\
\qquad set.v_n!(vv.n) \rightarrow \\
\qquad set.l_0!(vv.n+1) \rightarrow \cdots \rightarrow \\
\qquad set.l_m!(vv.m) \rightarrow \\
\qquad Skip
\end{array} \right)
\end{array} \right) \\
\textbf{provided} \\
\qquad \exists\, x_0 : \Gamma(v_0);\ \ldots;\ x_n : \Gamma(v_n);\ x_{n+1} : \Gamma(l_0);\ \ldots\ x_m : \Gamma(l_m) \bullet \\
\qquad\quad post(x_0, \ldots, x_n, x_{n+1}, \ldots, x_m) \\
\qquad\quad \wedge\ \overline{w}' = \overline{w} \\
\quad \textbf{where} \\
\qquad \overline{w}' = v'_0, \ldots, v'_n, l'_0, \ldots, l'_m \setminus w' \\
\qquad \overline{w} = v_0, \ldots, v_n, l_0, \ldots, l_m \setminus w
\end{array}
$$

In [Oli06], the semantics of assertions, coercions and (normalised) schema expressions is given in terms of specification statements. This is reflected in the definition of the rewriting function Ω_A of theses ***Circus*** constructs presented below.

$$\Omega_A(\{g\}) \mathrel{\widehat=} \Omega_A(: [g, true])$$

$$\Omega_A([g]) \mathrel{\widehat=} \Omega_A(: [g])$$

$$\Omega_A([udecl;\ ddecl' \mid pred]) = \Omega_A(ddecl : [\exists\, ddecl' \bullet pred, pred])$$

In ***Circus*** the renaming at the level of actions works on state components and local variables, not on channel names. For this reason, the rewriting of a renamed action is simply defined as the rewriting of the action resulting from the application of the rewriting.

$$\begin{aligned}&\Omega_A(A[old_1, \ldots, old_n := new_1, \ldots, new_n]) = \\ &\qquad \Omega_A(A[new_1, \ldots, new_n / old_1, \ldots, old_n])\end{aligned}$$

This concludes the definition of the rewriting function Ω_A for ***Circus*** actions.

The auxiliary function Ω'_A is very similar to Ω_A. It, however, does not read values from the memory and replaces references to variables by references to their local copies. For this reason, Ω'_A is the same as Ω_A for actions in which no values are retrieved like in $\Omega'_A(c \to A)$.

$$\Omega'_A(c \to A) \mathrel{\widehat=} c \to \Omega'_A(A)$$

For conciseness, we omit most of the definitions of Ω'_A, and present only those that differ from Ω_A.

In $c.e \to A$, the expression e might refer to memory components. The function Ω'_A, however, does not read them from the memory.

$$\begin{aligned}&\Omega'_A(c.e(v_0, \ldots, v_n, l_0, \ldots, l_m) \to A) \mathrel{\widehat=} \\ &\qquad c.e(vv_0, \ldots, vv_n, vl_0, \ldots, vl_m) \to \Omega'_A(A)\end{aligned}$$

The most important difference is for sequential compositions $A_1;\ A_2$: variables read in A_1 are not in the scope of A_2, which needs to access the memory again.

$$\Omega'_A(A_1;\ A_2) \mathrel{\widehat=} \Omega'_A(A_1);\ \Omega_A(A_2)$$

Syntactically, state updates in ***Circus*** can only be achieved by actions that require any subsequent action to be sequentially composed. The definition above guarantees that the rewritten version of A_2 reads the updated values before any reference to memory components.

Mapping *Circus* Stateless Actions into CSP Processes (Definition of Υ) The definition of the function Υ that maps *Circus* processes into CSP processes is extremely direct for most of the cases. In Appendix B we present the full definition of this translation function. Here, we focus on the most interesting parts of its definition.

For the vast majority of the *Circus* actions that are inherited from CSP, the translation is very straightforward. Among them, however, the restricted input prefixing $c?x : P \rightarrow A$ slightly differs from that of CSP. In *Circus*, the restriction P is given as a predicate on the state components and local variables, whereas in CSP this is given as a set. The mapping function Υ needs to take this difference into account: it returns a CSP prefixing restricted by the set we build based on the *Circus* predicate restriction.

$$\Upsilon\,(c?x : P \rightarrow A) \mathrel{\widehat=} \mathtt{c?x} : \{\mathtt{x} \mid \mathtt{x} \leftarrow \delta(c), \Upsilon_{\mathbb{B}}(P(x))\} \rightarrow \Upsilon(A)$$

Here, $\delta(c)$ returns the type of c and $\Upsilon_{\mathbb{B}}(P)$ is a specialisation of the mapping function for predicates, whose details are also presented in Appendix B.

The other non-trivial definition of the mapping function is that of alternation. In this mapping, we make use of a special event *choose*, which is used to guarantee that the non-deterministic choice is maintained in cases where more than one guard is valid. After retrieving all the variable values, the rewritten action offers a choice among those actions whose guards are valid, prefixed by the special event *choose*. If more than one guard is valid, as we are hiding this special event, the expected non-determinism takes place. If none of the guards are valid, as expected, the action diverges.

$$\begin{array}{l}
\Upsilon\left(\begin{array}{l}\textbf{if}\ g_0 \rightarrow\ A_0\\ \quad [\!]\ \ldots\\ \quad [\!]\ g_n \rightarrow\ A_n\\ \textbf{fi}\end{array}\right) \mathrel{\widehat=}\\
\quad\left(\begin{array}{l} g_0\ \&\ choose \rightarrow \Omega_A(A_0)\\ \Box \ldots\\ \Box\ g_n\ \&\ \ choose \rightarrow \Omega_A(A_n)\end{array}\right)\\
\quad \setminus \{\!|\ choose\ |\!\}\\
\textbf{provided}\ \bigvee g_i
\end{array}$$

Here, the proviso just reinforces that we are dealing with divergent free process; hence, at least one of the guards must be true.

In CSP, the **if** − **then** − **else** is available. Nevertheless, this construct is completely deterministic as it provides a sequence of conditional checking in nested alternations. The use of the solution above allows us to define Ω_A as an equality rather than a refinement, which would be the case if we had used CSP's **if** − **then** − **else** here.

As previously discussed, the correctness of both the mapping from ***Circus*** to CSP and the rewriting from ***Circus*** stateful actions to ***Circus*** stateless actions ought to be achieved in order to make our transformation strategy applicable. This is the subject of the next section.

5.3.2 Correctness

Proof of Correctness of Rewriting Function Ω The soundness of this transformation is achieved by induction on the syntax of ***Circus*** using the ***Circus*** refinement calculus presented in [Oli06]. For each element in the ***Circus*** syntax, we demonstrate that the refinement of Figure 9 is valid.

By way of illustration, we present below the proof for transforming a process with *Skip* as its main action. In what follows, $P_S.A$ denotes a process named P whose state is S and main action is A.

Theorem K.1

$$\begin{array}{l} P_S.\mathit{Skip} \\ = \\ \Omega(P_S.\mathit{Skip}) \end{array}$$

Proof. Starting from the right-hand side, we start the proof by simply applying the definition of Ω and Ω_A.

$$\begin{array}{ll}
\Omega(P_S.\mathit{Skip}) & [\Omega] \\
= P.\mathbf{var}\ b : \left\{ \begin{array}{l} x : \mathit{BINDING} \mid b(v_0) \in T_0 \wedge \ldots \\ \qquad\qquad\qquad \wedge\ \mathit{inv}(b(v_0), \ldots, b(v_n)) \end{array} \right\} \bullet & \\
\qquad \left(\begin{array}{l} (\Omega_A(\mathit{Skip});\ \mathit{terminate} \rightarrow \mathit{Skip}) \\ [\![\emptyset \mid \mathit{MEM}_I \mid \emptyset]\!] \\ \mathit{Memory}(b) \end{array} \right) \setminus \mathit{MEM}_I &
\end{array}$$

[Ω_A]

$$
\begin{array}{l}
= P.\mathbf{var}\ b : \left\{ \begin{array}{l} x : \mathit{BINDING} \mid b(v_0) \in T_0 \land \dots \\ \hfill \land\ \mathit{inv}(b(v_0), \dots, b(v_n)) \end{array} \right\} \bullet \\
\left(\begin{array}{l} (\mathit{Skip};\ \mathit{terminate} \rightarrow \mathit{Skip}) \\ \lbrack\!\lbrack \emptyset \mid \mathit{MEM}_I \mid \{b\} \rbrack\!\rbrack \\ \mathit{Memory}(b) \end{array} \right) \setminus \mathit{MEM}_I
\end{array}
$$

Next, the application of the refinement law of sequence unit removes the *Skip* action.

[Law 8]

$$
\begin{array}{l}
= P.\mathbf{var}\ b : \left\{ \begin{array}{l} x : \mathit{BINDING} \mid b(v_0) \in T_0 \land \dots \\ \hfill \land\ \mathit{inv}(b(v_0), \dots, b(v_n)) \end{array} \right\} \bullet \\
\left(\begin{array}{l} (\mathit{terminate} \rightarrow \mathit{Skip}) \\ \lbrack\!\lbrack \emptyset \mid \mathit{MEM}_I \mid \{b\} \rbrack\!\rbrack \\ \mathit{Memory}(b) \end{array} \right) \setminus \mathit{MEM}_I
\end{array}
$$

We then unfold the recursive action *Memory*.

[Law 9]

$$
\begin{array}{l}
= P.\mathbf{var}\ b : \left\{ \begin{array}{l} x : \mathit{BINDING} \mid b(v_0) \in T_0 \land \dots \\ \hfill \land\ \mathit{inv}(b(v_0), \dots, b(v_n)) \end{array} \right\} \bullet \\
\left(\begin{array}{l} (\mathit{terminate} \rightarrow \mathit{Skip}) \\ \lbrack\!\lbrack \emptyset \mid \mathit{MEM}_I \mid \{b\} \rbrack\!\rbrack \\
\left(\begin{array}{l} \mathbf{vres}\ b : \mathit{BINDING} \bullet \\
\qquad \left(\begin{array}{l} \Box\, n : \mathit{NAME} \bullet \\ \qquad \mathit{get}.n!b(n) \rightarrow \mathit{Cell}(b) \end{array} \right) \\
\qquad \Box \left(\begin{array}{l} \Box\, n : \mathit{NAME} \bullet \\ \qquad \mathit{set}.n?\mathit{nv} \rightarrow \\ \qquad \mathit{Cell}(b \oplus \{n \mapsto \mathit{nv}\}) \end{array} \right) \\
\qquad \Box\ \mathit{terminate} \rightarrow \mathit{Skip} \end{array} \right) \\
\quad (b) \end{array} \right) \setminus \mathit{MEM}_I
\end{array}
$$

Our intention is to expand the definition of *Memory*. In order to avoid conflicts in variable names, we rename the outermost b.

[Law 49]

$$
\begin{array}{l}
= P.\mathbf{var}\ sb : \left\{ \begin{array}{l} x : \mathit{BINDING} \mid sb(v_0) \in T_0 \wedge \ldots \\ \qquad\qquad\qquad\qquad \wedge\ \mathit{inv}(sb(v_0), \ldots, sb(v_n)) \end{array} \right\} \bullet \\
\quad \left(\begin{array}{l}
(\mathit{terminate} \to \mathit{Skip}) \\
[\![\emptyset \mid \mathit{MEM}_I \mid \{sb\}]\!] \\
\left(\begin{array}{l}
\mathbf{vres}\ b : \mathit{BINDING} \bullet \\
\qquad \left(\begin{array}{l} \Box\, n : \mathit{NAME} \bullet \\ \qquad \mathit{get}.n!b(n) \to \mathit{Cell}(b) \end{array} \right) \\
\qquad \Box \left(\begin{array}{l} \Box\, n : \mathit{NAME} \bullet \\ \qquad \mathit{set}.n?nv \to \\ \qquad \mathit{Cell}(b \oplus \{n \mapsto nv\}) \end{array} \right) \\
\qquad \Box\ \mathit{terminate} \to \mathit{Skip}
\end{array} \right) \\
\quad (sb)
\end{array} \right) \setminus \mathit{MEM}_I
\end{array}
$$

We may now use the semantics of **vres**.

[Semantics of **vres**]

$$
\begin{array}{l}
= P.\mathbf{var}\ sb : \left\{ \begin{array}{l} x : \mathit{BINDING} \mid sb(v_0) \in T_0 \wedge \ldots \\ \qquad\qquad\qquad\qquad \wedge\ \mathit{inv}(sb(v_0), \ldots, sb(v_n)) \end{array} \right\} \bullet \\
\quad \left(\begin{array}{l}
(\mathit{terminate} \to \mathit{Skip}) \\
[\![\emptyset \mid \mathit{MEM}_I \mid \{sb\}]\!] \\
\left(\begin{array}{l}
\mathbf{var}\ b : \mathit{BINDING} \bullet \\
\qquad b := sb; \\
\qquad \left(\begin{array}{l}
\left(\begin{array}{l} \Box\, n : \mathit{NAME} \bullet \\ \qquad \mathit{get}.n!b(n) \to \mathit{Cell}(b) \end{array} \right) \\
\Box \left(\begin{array}{l} \Box\, n : \mathit{NAME} \bullet \\ \qquad \mathit{set}.n?nv \to \\ \qquad \mathit{Cell}(b \oplus \{n \mapsto nv\}) \end{array} \right) \\
\Box\ \mathit{terminate} \to \mathit{Skip}
\end{array} \right); \\
\qquad sb := b
\end{array} \right)
\end{array} \right) \\
\quad \setminus \mathit{MEM}_I
\end{array}
$$

The local variable b is not referenced in the left-hand side of the parallel composition. Hence, this variable block may be expanded.

[Law 1]

$$
\begin{array}{l}
= P.\mathbf{var}\ sb : \left\{ \begin{array}{l} x : BINDING \mid sb(v_0) \in T_0 \land \ldots \\ \qquad\qquad\qquad\qquad \land\ inv(sb(v_0), \ldots, sb(v_n)) \end{array} \right\} \bullet \\
\quad \mathbf{var}\ b : BINDING \bullet \\
\qquad \left(\begin{array}{l}
(terminate \rightarrow Skip) \\
[\![\emptyset \mid MEM_I \mid \{sb\}]\!] \\
\left(\begin{array}{l}
b := sb; \\
\left(\begin{array}{l}
\left(\begin{array}{l} \Box\, n : NAME \bullet \\ \qquad get.n!b(n) \rightarrow Cell(b) \end{array} \right) \\
\Box \left(\begin{array}{l} \Box\, n : NAME \bullet \\ \qquad set.n?nv \rightarrow \\ \qquad Cell(b \oplus \{n \mapsto nv\}) \end{array} \right) \\
\Box\ terminate \rightarrow Skip
\end{array} \right); \\
sb := b
\end{array} \right)
\end{array} \right) \\
\qquad \setminus MEM_I
\end{array}
$$

Next, we may use the Law 47 to remove the first assignment.

[Law 47]

$$
\begin{array}{l}
= P.\mathbf{var}\ sb : \left\{ \begin{array}{l} x : BINDING \mid sb(v_0) \in T_0 \land \ldots \\ \qquad\qquad\qquad\qquad \land\ inv(sb(v_0), \ldots, sb(v_n)) \end{array} \right\} \bullet \\
\quad \mathbf{var}\ b : BINDING \bullet \\
\qquad \left(\begin{array}{l}
(terminate \rightarrow Skip) \\
[\![\emptyset \mid MEM_I \mid \{sb\}]\!] \\
\left(\begin{array}{l}
\left(\begin{array}{l}
\left(\begin{array}{l} \Box\, n : NAME \bullet \\ \qquad get.n!sb(n) \rightarrow Cell(b) \end{array} \right) \\
\Box \left(\begin{array}{l} \Box\, n : NAME \bullet \\ \qquad set.n?nv \rightarrow \\ \qquad Cell(sb \oplus \{n \mapsto nv\}) \end{array} \right) \\
\Box\ terminate \rightarrow Skip
\end{array} \right); \\
sb := sb
\end{array} \right)
\end{array} \right) \\
\qquad \setminus MEM_I
\end{array}
$$

When replacing b for sb, we are left with an innocuous assignment, which can be removed as follows.

[Laws 48 and 8]

$$
\begin{array}{l}
= P.\mathbf{var}\ sb : \left\{ \begin{array}{l} x : BINDING \mid sb(v_0) \in T_0 \land \ldots \\ \qquad\qquad\qquad\qquad \land\ inv(sb(v_0), \ldots, sb(v_n)) \end{array} \right\} \bullet \\
\quad \mathbf{var}\ b : BINDING \bullet \\
\qquad \left(\begin{array}{l} (terminate \to Skip) \\ [\![\emptyset \mid MEM_I \mid \{sb\}]\!] \\ \left(\begin{array}{l} \left(\begin{array}{l} \Box\, n : NAME \bullet \\ \quad get.n!sb(n) \to Cell(b) \end{array} \right) \\ \Box \left(\begin{array}{l} \Box\, n : NAME \bullet \\ \quad set.n?nv \to \\ \quad Cell(sb \oplus \{n \mapsto nv\}) \end{array} \right) \\ \Box\ terminate \to Skip \end{array} \right) \end{array} \right) \setminus MEM_I
\end{array}
$$

Next, since the variable b is no longer referenced, the variable block may be removed.

[Laws 6]

$$
\begin{array}{l}
= P.\mathbf{var}\ sb : \left\{ \begin{array}{l} x : BINDING \mid sb(v_0) \in T_0 \land \ldots \\ \qquad\qquad\qquad\qquad \land\ inv(sb(v_0), \ldots, sb(v_n)) \end{array} \right\} \bullet \\
\qquad \left(\begin{array}{l} (terminate \to Skip) \\ [\![\emptyset \mid MEM_I \mid \{sb\}]\!] \\ \left(\begin{array}{l} \left(\begin{array}{l} \Box\, n : NAME \bullet \\ \quad get.n!sb(n) \to Cell(b) \end{array} \right) \\ \Box \left(\begin{array}{l} \Box\, n : NAME \bullet \\ \quad set.n?nv \to \\ \quad Cell(sb \oplus \{n \mapsto nv\}) \end{array} \right) \\ \Box\ terminate \to Skip \end{array} \right) \end{array} \right) \setminus MEM_I
\end{array}
$$

Just for the sake of naming conventions, we rename the outermost variable

sb back to b.

[Law 49]

$$
\begin{array}{l}
= P.\mathbf{var}\ b : \left\{ \begin{array}{l} x : BINDING \mid b(v_0) \in T_0 \wedge \ldots \\ \qquad\qquad\qquad \wedge\ inv(b(v_0), \ldots, b(v_n)) \end{array} \right\} \bullet \\
\left(\begin{array}{l}
(terminate \rightarrow Skip) \\
[\![\emptyset \mid MEM_I \mid \{b\}]\!] \\
\left(\begin{array}{l}
\left(\begin{array}{l} \Box\, n : NAME \bullet \\ \quad get.n!b(n) \rightarrow Cell(b) \end{array} \right) \\
\Box \left(\begin{array}{l} \Box\, n : NAME \bullet \\ \quad set.n?nv \rightarrow \\ \quad Cell(b \oplus \{n \mapsto nv\}) \end{array} \right) \\
\Box\ terminate \rightarrow Skip
\end{array} \right)
\end{array} \right) \setminus MEM_I
\end{array}
$$

Because all channels are in the synchronisation channel set, the only possible synchronisation is on *terminate*. We use the Law 10 to remove the external choice in the parallel composition.

[Law 10]

provided

$\{terminate\} \subseteq MEM_I$

$\{get, set\} \subseteq MEM_I$

$\{get, set\} \cap \{terminate\} = \emptyset$

$$
\begin{array}{l}
= P.\mathbf{var}\ b : \left\{ \begin{array}{l} x : BINDING \mid b(v_0) \in T_0 \wedge \ldots \\ \qquad\qquad\qquad \wedge\ inv(b(v_0), \ldots, b(v_n)) \end{array} \right\} \bullet \\
\left(\begin{array}{l}
(terminate \rightarrow Skip) \\
[\![\emptyset \mid MEM_I \mid \{b\}]\!] \\
(terminate \rightarrow Skip)
\end{array} \right) \setminus MEM_I
\end{array}
$$

Next, since the synchronisation on *terminate* is the only option and hidden from the environment, we may ignore it.

[Law 25]

provided

$[terminate \in MEM_I]$

$$
\begin{array}{l}
= P.\mathbf{var}\ b : \left\{ \begin{array}{l} x : BINDING \mid b(v_0) \in T_0 \wedge \ldots \\ \qquad\qquad\qquad \wedge\ inv(b(v_0), \ldots, b(v_n)) \end{array} \right\} \bullet \\
\left(\begin{array}{l}
Skip \\
[\![\emptyset \mid MEM_I \mid \{b\}]\!] \\
Skip
\end{array} \right) \setminus MEM_I
\end{array}
$$

We are left with the parallel composition on *Skip*, which may be removed using the unit law for parallel composition.

[Law 28]

provided

$[terminate \in MEM_I]$

$$= P.\mathbf{var}\ b : \left\{ \begin{array}{l} x : BINDING \mid b(v_0) \in T_0 \land \ldots \\ \qquad\qquad\qquad \land\ inv(b(v_0), \ldots, b(v_n)) \end{array} \right\} \bullet \quad Skip \setminus MEM_I$$

The hiding may be ignore because it has no effect on *Skip*.

[Law 15]

provided

$[MEM_I \cap usedC(Skip) = \emptyset]$

$$= P.\mathbf{var}\ b : \left\{ \begin{array}{l} x : BINDING \mid b(v_0) \in T_0 \land \ldots \\ \qquad\qquad\qquad \land\ inv(b(v_0), \ldots, b(v_n)) \end{array} \right\} \bullet \quad Skip$$

Finally, we promote the variable b to a state component of a stateless process. This concludes the proof since we are left with the left-hand side of the theorem.

[Law 26(b is the only component of S)]

$$= P_S.Skip$$

$\square$

Proofs of other *Circus* actions can be found in Appendix K.

Proof of Correctness of the Mapping Function Υ. In [CG10], Cavalcanti *et al* present the following definitions that provide a link between *Circus* and CSP theories within the UTP.

The predicate A^n defined below gives the behaviour of the action A when its preceding action has not diverged and has terminated, and when A itself does not lead to divergence.

$$A^n \mathrel{\widehat{=}} okay \land \neg\, wait \land A \land okay'$$

This is the normal behaviour of A; behaviour in other situations is defined by healthiness conditions.

The terminating, non-diverging behaviour of A is A^t as presented below.

$$A^t \mathrel{\widehat=} A^n \land \lnot\, wait'$$

Finally, the diverging behaviour of A is

$$A^d \mathrel{\widehat=} okay \land \lnot\, wait \land A \land \lnot\, okay'.$$

The function $traces^{\mathcal{UTP}}$ defined in [CG10] and presented below gives the set of traces of a ***Circus*** action defined as a UTP predicate A. This gives a traces model to A compatible with that adopted in the failures-divergences model of CSP.

As already said, the behaviour of the action itself is that prescribed when $okay$ and $\lnot\, wait$. The behaviour in the other cases is determined by healthiness conditions of the UTP theory. For example, in the presence of divergence, that is, when $\lnot\, okay$, every action can only guarantee that the trace is only extended, so that past history is not modified. This behaviour is not recorded by $traces^{\mathcal{UTP}}(A)$.

$$traces^{\mathcal{UTP}}(A) = \{tr' - tr \mid A^n\} \cup \{(tr' - tr) \frown \langle \checkmark \rangle \mid A^t\}$$

As mentioned in Section 2.4, tr records the history of interactions before the start of the action; tr'carries this history forward. Therefore, the traces in $traces^{\mathcal{UTP}}(A)$ are sequences $tr' - tr$ obtained by removing from tr' its prefix tr. In addition, if $tr' - tr$ leads to termination, then $traces^{\mathcal{UTP}}(A)$ also includes $(tr' - tr) \frown \langle \checkmark \rangle$, since ✓ is used in the failures-divergences model to signal termination.

In this document, we follow the syntactic sugaring used in [CG10] to express set comprehension. In this notation, we implicitly have an outermost set comprehension which existentially quantifies all UTP observational variables and their dashed counterparts. For instance, in the definition of $traces^{\mathcal{UTP}}$ above we write $\{tr' - tr \mid A^n\}$ as the set of all traces executed by the process in which the condition A^n is satisfied. Strictly speaking, this set comprehension contains tr and tr' as free-variables. However, this notation is used in [CG10] to denote the Z set-comprehension $\{tr, tr : \mathrm{seq}(EVENT) \mid A^n \bullet tr' - tr\}$, hence, tr and tr' are actually not free-variables but existentially quantified.

The $divergences^{\mathcal{UTP}}$ are those traces that lead the action to divergence.

$$divergences^{\mathcal{UTP}}(A) = \{tr' - tr \mid A^d\}$$

In [CG10], the authors have actually introduced the set $\mathit{traces}_{\bot}^{\mathcal{UTP}}(A)$, which is defined as follows to include all traces that lead to divergence.

$$\mathit{traces}_{\bot}^{\mathcal{UTP}}(A) = \mathit{traces}^{\mathcal{UTP}}(A) \cup \mathit{divergences}^{\mathcal{UTP}}(A)$$

The function defined below gives the set of failures of a divergence-free action A.

$$\begin{aligned}\mathit{failures}^{\mathcal{UTP}}(A) = {} & \{(tr' - tr, ref') \mid A^n\} \\ & \cup \{(tr' - tr, ref' \cup \{\checkmark\}) \mid A^n \wedge wait'\} \\ & \cup \{((tr' - tr) \frown \langle \checkmark \rangle, ref') \mid A^t\} \\ & \cup \{((tr' - tr) \frown \langle \checkmark \rangle, ref' \cup \{\checkmark\}) \mid A^t\}\end{aligned}$$

In a state that is not terminating, for every refusal set ref', there is an extra set $ref' \cup \{\checkmark\}$. This is because $\checkmark$ is not part of the UTP model and is not considered in the definition of ref', just as it is not considered in the definition of tr'. As before, for a terminating state, the extra trace $(tr' - tr) \frown \langle \checkmark \rangle$ is recorded. Finally, after termination, $\checkmark$ is also refused, and so $ref' \cup \{\checkmark\}$ is included.

The following definition was not in [CG10], but it is based on a similar definition from [CW06]; it includes all refusals in the presence of divergence.

$$\begin{aligned}& \mathit{failures}_{\bot}^{\mathcal{UTP}}(A) = \\ & \quad \mathit{failures}^{\mathcal{UTP}}(A) \cup \{(s, ref) \mid s \in \mathit{divergences}^{\mathcal{UTP}}(P) \wedge ref \in \Sigma^{*\checkmark}\}\end{aligned}$$

Furthermore, in [CW06], traces and failures refinement are defined in the expected way as presented below.

$$\begin{aligned}P \sqsubseteq_{\mathrm{T}}{}^{\mathcal{UTP}}\ Q & \Leftrightarrow \mathit{traces}^{\mathcal{UTP}}(Q) \subseteq \mathit{traces}^{\mathcal{UTP}}(P) \\ P \sqsubseteq_{\mathrm{F}}{}^{\mathcal{UTP}}\ Q & \Leftrightarrow \mathit{traces}^{\mathcal{UTP}}(Q) \subseteq \mathit{traces}^{\mathcal{UTP}}(P) \\ & \quad \wedge \mathit{failures}^{\mathcal{UTP}}(Q) \subseteq \mathit{failures}^{\mathcal{UTP}}(P)\end{aligned}$$

Since we are dealing with divergent free processes we have that:

$$\begin{aligned}& \quad P \sqsubseteq_{\mathrm{FD}}{}^{\mathcal{UTP}}\ Q \\ \Leftrightarrow & \\ & \quad \mathit{failures}_{\bot}^{\mathcal{UTP}}(Q) \subseteq \mathit{failures}_{\bot}^{\mathcal{UTP}}(P) \\ & \quad \wedge \mathit{divergences}^{\mathcal{UTP}}(Q) \subseteq \mathit{divergences}^{\mathcal{UTP}}(Q) \\ \Leftrightarrow & \\ & \quad P \sqsubseteq_{\mathrm{F}}{}^{\mathcal{UTP}}\ Q\end{aligned}$$

In [CG07, CG10], Cavalcanti *et al* demonstrated that provided P_1 and P_2 are divergence-free ***Circus*** processes with main actions A_1 and A_2, we can characterise refinement as follows.

- $P_1 \sqsubseteq_{\mathrm{P}} P_2 \Leftrightarrow A_1 \sqsubseteq_{\mathrm{T}}{}^{\mathcal{UTP}} A_2 \wedge A_2 \, conf \, A_1$ (from [CG10])
- $A_1 \sqsubseteq_{\mathrm{F}}{}^{\mathcal{UTP}} A_2 \Leftrightarrow A_1 \sqsubseteq_{\mathrm{T}}{}^{\mathcal{UTP}} A_2 \wedge A_2 \, conf \, A_1$ (from [CG07])

where

$$A_2 \, conf \, A_1 \mathrel{\widehat=} \forall t : traces(A_1) \cap traces(A_2) \bullet Ref(A_2, t) \subseteq Ref(A_1, t)$$

$$Ref(A, t) \mathrel{\widehat=} \{X \mid (t, X) \in failures(A)\}$$

For non-divergent processes $\mathcal{F}$ refinement corresponds to $\mathcal{FD}$ refinement. So, previous results guarantee that, for non-divergent processes:

- $P_1 \sqsubseteq_{\mathrm{P}} P_2 \Leftrightarrow A_1 \sqsubseteq_{\mathrm{F}}{}^{\mathcal{UTP}} A_2$
- $P_1 \sqsubseteq_{\mathrm{P}} P_2 \Leftrightarrow A_1 \sqsubseteq_{\mathrm{FD}}{}^{\mathcal{UTP}} A_2$

Based on these results on linking the semantic domains, we establish the correctness of a translation Υ, that maps ***Circus*** processes into CSP processes. The soundness of this mapping from ***Circus*** to CSP is established for the traces and the failures models.

The proof of this theorem is achieved by induction on the syntax of ***Circus*** that has a corresponding action in CSP (such actions are in the domain ***Circus***$_{CSP}$ of Υ, presented in Appendix B). For every ***Circus*** action A that it is mapped into a CSP process P we prove that the set of traces of A generated by the UTP function is equal to the set of traces of P in CSP as defined in [Ros98].

Theorem 5.1 *For every* ***Circus*** *process P in* dom(Υ)

$$traces^{\mathcal{UTP}}(P) = traces(\Upsilon(P))$$

where *traces* is the original traces CSP semantic function as defined in [Ros98]. We do the same for the failures model.

Theorem 5.2 *For every* ***Circus*** *process P in* dom(Υ)

$$failures^{\mathcal{UTP}}(P) = failures(\Upsilon(P))$$

where *failures* is the original failures CSP semantic function as defined in [Ros98].

By way of illustration, we present below the first part of the proof for the *Skip* action, in which we prove that the traces of the UTP semantics of the ***Circus*** *Skip* is the same as the traces of the CSP `SKIP`.

Theorem J.2 $traces^{\mathcal{UTP}}(Skip) = traces(\Upsilon(Skip))$

Here, we make use A^b_c to denote $A[b/okay'][c/wait]$. Furthermore, we take into consideration that the special event ✓ used in [Ros98] is not allowed in the UTP traces tr and tr'.

Proof. Starting from the left-hand side, we start the proof by simply applying the definition of the UTP traces.

$$\begin{array}{ll} traces^{\mathcal{UTP}}(Skip) & [traces^{\mathcal{UTP}}] \\ = \{tr' - tr \mid (Skip)^n\} & [A^t] \\ \quad \cup \{(tr' - tr) \frown \langle ✓ \rangle \mid (Skip)^t\} & \end{array}$$

Next, the behaviour of a terminating action is defined in terms of its normal behaviour.

$$\begin{array}{ll} = \{tr' - tr \mid (Skip)^n\} & [A^n] \\ \quad \cup \{(tr' - tr) \frown \langle ✓ \rangle \mid \neg\, wait' \wedge (Skip)^n\} & \end{array}$$

The normal behaviour of an action corresponds to those situations in which the action stars in a non-divergent state ($okay \wedge \neg\ wait$) and does not diverge ($okay'$).

$$\begin{array}{ll} = \{tr' - tr \mid okay \wedge \neg\, wait \wedge okay' \wedge Skip\} & [\text{PC}] \\ \quad \cup \{(tr' - tr) \frown \langle ✓ \rangle \mid okay \wedge \neg\, wait \wedge okay' \wedge \neg\, wait' \wedge Skip\} & \end{array}$$

Using the predicate calculus, we may transform the predicate in order to use the same notation as in [Oli06].

$$\begin{array}{ll} = \{tr' - tr \mid okay \wedge (Skip)^t_f\} & [\text{Lemma J.12}] \\ \quad \cup \{(tr' - tr) \frown \langle ✓ \rangle \mid okay \wedge \neg\, wait' \wedge (Skip)^t_f\} & \end{array}$$

Now, we may use a theorem proved in [Oli06], which gives the behaviour of the *Skip* action when it is not waiting and does not diverge.

$$\begin{array}{ll} = \{tr' - tr \mid okay \wedge \mathbf{CSP1}(tr' = tr \wedge \neg\, wait' \wedge v' = v)\} & [\text{Lemma J.4}] \\ \quad \cup \left\{ \begin{array}{l} (tr' - tr) \frown \langle ✓ \rangle \\ \mid okay \wedge \neg\, wait' \wedge \mathbf{CSP1}(tr' = tr \wedge \neg\, wait' \wedge v' = v) \end{array} \right\} & \end{array}$$

A **CSP1** action that starts on a divergent state ($\neg\ okay$) is only guaranteed not the forget the traces ($\mathbf{CSP1}(A) \mathrel{\widehat=} (\neg\ okay \wedge tr \leq tr') \vee A$). Nevertheless, in our case, we have *okay* in the context; hence, **CSP1** might be ignored.

$$\begin{array}{ll} = \{tr' - tr \mid okay \wedge tr' = tr \wedge \neg\, wait' \wedge v' = v\} & [\text{SS. and} -] \\ \quad \cup \{(tr' - tr) \frown \langle ✓ \rangle \mid okay \wedge \neg\, wait' \wedge tr' = tr \wedge \neg\, wait' \wedge v' = v\} & \end{array}$$

In both set comprehensions, we have that $tr' = tr$ must be satisfied. Hence, the sequence subtraction yields, in both cases, an empty sequence.

$$\begin{aligned} &= \{\langle\rangle \mid okay \land tr' = tr \land \neg\, wait' \land v' = v\} && \text{[Cases and SC]} \\ &\quad \cup \{\langle \checkmark \rangle \mid okay \land tr' = tr \land \neg\, wait' \land v' = v\} \end{aligned}$$

Finally, a simple case analysis on the boolean conditions gives us the resulting set below since we have a constant sequence in the production of the set comprehension. Here, it is important to remind readers of the syntactic sugaring we inherited from [CG10]. Hence, the set comprehension above contains no free-variables since all UTP observational variables are implicitly existentially quantified.

$$= \{\langle\rangle, \langle \checkmark \rangle\} \qquad [traces]$$

This corresponds exactly to the semantic definition of `SKIP` in the traces model of [Ros98].

$$= traces(\mathtt{SKIP}) \qquad [\Upsilon]$$

The definition of the Υ function concludes this proof..

$$= traces(\Upsilon(\mathtt{Skip}))$$

$\square$

The same proof strategy is used for proving the correctness of the mapping of *Skip* in the *failures* model. Proofs of other ***Circus*** constructs that are directly mapped into CSP can be found in Appendix J.

With this proof, based on [CG07, CG10], we are able to establish a connection between traces refinement and failures refinement in the UTP and CSP.

- $P \sqsubseteq_{\mathrm{T}} Q \quad \Leftrightarrow \quad P \sqsubseteq_{\mathrm{T}}^{\mathcal{UTP}} Q$
- $P \sqsubseteq_{\mathrm{F}} Q \quad \Leftrightarrow \quad P \sqsubseteq_{\mathrm{F}}^{\mathcal{UTP}} Q$

As previously said, we consider only divergent free processes. For this reason, we take only the $\mathcal{T}$ and $\mathcal{F}$ models into consideration. For divergent free processes, we have that the latter is equivalent to $\mathcal{FD}$.

5.4 From CML to *Circus*

As already indicated, ***Circus*** and CML are similar languages in many ways. First of all, they are both based on a language for data-modelling and CSP.

In the case of *Circus* the data language is Z, and in the case of CML, it is VDM. In addition, CML also includes constructs for object orientation based on VDM++ and an object-oriented extension of *Circus* [CSW05b], and constructs for time modelling based on Timed CSP and a timed extension of *Circus* [SCHS10]. When, however, we restrict CML to the subset currently considered in this work, namely the untimed language without object-orientation, the correspondence with *Circus* is rather direct. It is strengthened by the fact that both *Circus* and CML have a semantic model based on the UTP, and an extra assumption: the CML models does not have any undefined expression. In this section, we discuss the translation from CML to *Circus*, so that the results previously discussed can be applied to CML specifications.

5.4.1 Mapping CML into *Circus*

All in all, untimed CML models without object-oriented constructs and without undefined expressions can be translated to *Circus* using a mapping function ρ. Given the closeness of the two notations, it is a simple function that distributes through the structure of processes and actions. Table 3 describes how the process operators of CML, as defined in [WCC+12], map to those of *Circus*. We use different fonts to distinguish elements of the CML syntax from the corresponding element obtained by applying ρ. For example, if P is a process described using the CML (ASCII-based) syntax, which is supported by the COMPASS tools, the corresponding *Circus* process $\rho(\mathrm{P})$ is written P.

Given the close relationship between the process notations of CML and *Circus*, it is not surprising that the translation is simple. Table 3 does not give the details of the translation of declarations d and channel set expressions cse; this is a rather simple matter. For expressions in general, the main issue is the possibility of undefined expressions, which are treated in very different ways in VDM and Z. For models without such expressions, the translation is just a syntactic issue as well, as explained and proved in the COMPASS Deliverable [BBC+12]. We omit the definition of the translation of the CML replicated operators, which are also available in *Circus*.

We use $Stop$ to denote the *Circus* stateless process with main action $Stop$. We use that to define alphabetised parallelism in terms of generalised parallelism. This is a standard definition, that uses the set Σ of all channels in scope.

Circus does not have renaming comprehensions or renaming of events. In

CML Process	*Circus* Process (ρ)
‘`begin`’, [‘`state`’, d], ‘`@`’, A, ‘`end`’	**begin** [**state** d] $\bullet$ A **end**
P1, ‘`;`’, P2	$P_1;P_2$
P, ‘`[]`’, P	$P_1 \Box P_2$
P1, ‘`|~|`’, P2	$P_1 \sqcap P_2$
P2, ‘`[|`’, cse, ‘`|]`’, P2	$P_1 \lbrack\!\lbrack\, cse \,\rbrack\!\rbrack\, P_2$
P1, ‘`[`’, cse1, ‘`||`’, cse2, ‘`]`’, P2	$(P_1 \lbrack\!\lbrack\, \Sigma \setminus cse_1 \,\rbrack\!\rbrack\, Stop)$ $\lbrack\!\lbrack cse_1 \cap cse_2 \rbrack\!\rbrack$ $(P_2 \lbrack\!\lbrack\, \Sigma \setminus cse_2 \,\rbrack\!\rbrack\, Stop)$
P1 , ‘`||`’, P2	$P_1 \lbrack\!\lbrack\, \Sigma \,\rbrack\!\rbrack\, P_2$
P1, ‘`|||`’, P2	$P_1 \interleave P_2$
P, ‘`\`’, cse	$P \setminus cse$
‘`(`’, d, ‘`@`’, P, ‘`)`’, ‘`(`’, { E }, ‘`)`’	$(d \bullet P)(E)$
N, [‘`(`’,{ E }, ‘`)`’]	$N(E)$
P ‘`[[`’, N1 ‘`<-`’ N2, ‘`]]`’	$P[N_1 := N_2]$

Table 3: Mapping between CML and *Circus* process operators
.

Circus, renaming is only available for channels, rather than individual events. This reflects the view that channels (rather than events) are used to model interaction points that have a direct correspondence to elements of the programs or systems described. This means that the use of channels can be implemented in a direct way. Only when multi-synchronisation is required, protocols are usually necessary to generate concrete systems. CML takes the more abstract view of CSP. In this case, a channel potentially models an array or matrix of interaction points, which can be used independently.

To translate CML models that take advantage of such facility, we require a (non-compositional) pre-processing before the application of ρ. It is necessary to identify all events used as an independent channel: an individual target of a renaming, or in a channel set expression that does not necessarily include all other events on the same channel. For each of these, we need to declare a new channel, and rewrite the whole CML model to use the new channel instead of the event. We, therefore, rule out this facility of CML.

Tables 4, 5, and 6 define ρ for actions. Since the language of processes (of both CML and *Circus*) is just a lift of some of the action operators to the component and system level of processes, some of the translations are rather similar to those in Table 3, and equally simple. We again do not include the replicated operators, which are also available in *Circus*. The same issues related to renaming arise for actions as well.

CML includes versions of the *Circus* action parallelism operators without name set expressions. For these, the name sets are assumed to be empty. Translation of name-set expressions nse is trivial.

Like with *Circus* processes, without loss of generality, we are considering CML processes whose all local action definitions are removed (using the copy rule). The let action constructor, on the other hand, introduces local definitions of its own, and has no equivalent in *Circus*. Translation, therefore, requires that all occurrences of let are removed simply by flattening the local scopes, and later removing all auxiliary action definitions so introduced.

Multiple assignments are listed individually in CML as atomic assignments. We need to collect all assigned variables and assigning expressions together to form two lists and compose a *Circus* assignment. In Table 5 we show the result for a multiple assignment with two components. In the translation of an implicit operation body, the function δ applies to a frame f and extracts the list of variables that can be changed: those associated with a mode wr.

Circus includes the nondeterministic conditional of CML. Table 5 shows the translation of conditionals with two guards. Similarly, to define the

CML Action	*Circus* Action (ρ)
‘Skip’	$Skip$
‘Stop’	$Stop$
‘Chaos’	$(\mu X \bullet \Box c : \Sigma \bullet c?x \to X) \sqcap Stop$
‘Div’	$Chaos$
c, ‘->’, A	$c \to A$
E, ‘&’, A	E & A
A1, ‘;’, A2	$A_1 ; A_2$
A1, ‘[]’, A2	$A_1 \Box A_2$
A1, ‘|~|’, A2	$A_1 \sqcap A_2$
A1, ‘\’, cse	$A_1 \setminus cse$
A ‘[[’, N1 ‘<-’ N2, ‘]]’	$A[N_1 := N_2]$
‘mu’, N, ‘@’, A	$\mu N \bullet A$
A1, ‘[||’, nse1, ‘|’, nse2, ‘||]’, A2	$A_1 \parallel[nse_1 \mid nse_2]\parallel A_2$
A1, ‘|||’, A2	$A_1 \parallel[\{\} \mid \{\}]\parallel A_2$
A1‘[|’, nse1, ‘|’, ns2, ‘|]’, A2	$A_1 \llbracket nse_1 \mid \Sigma \mid nse_2 \rrbracket A_2$
A1 ‘||’ A2,	$A_1 \llbracket \{\} \mid \Sigma \mid \{\} \rrbracket A_2$
A1 ‘[’, nse1, ‘|’, cse1, ‘||’, cse2, ‘|’, nse2, ‘]’, A2	$(A_1 \llbracket nse_1 \mid \Sigma \setminus cse_1 \mid \{\} \rrbracket Stop)$ $\parallel[nse_1 \mid cse_1 \cap cse_2 \mid nse_2]\parallel$ $(A_2 \llbracket nse_2 \mid \Sigma \setminus cse_2 \mid \{\} \rrbracket Stop)$
A1 ‘[’, cse1, ‘||’, cse2, ‘]’, A2	$(A_1 \llbracket \{\} \mid \Sigma \setminus cse_1 \mid \{\} \rrbracket Stop)$ $\parallel[\{\} \mid cse_1 \cap cse_2 \mid \{\}]\parallel$ $(A_2 \llbracket \{\} \mid \Sigma \setminus cse_2 \mid \{\} \rrbracket Stop)$
A1 ‘[|’, nse1, ‘|’, cse, ‘|’, nse2, ‘|]’, A2	$A_1 \llbracket nse_1 \mid cse \mid nse_2 \rrbracket A_2$
A1 ‘[|’, cse, ‘|]’, A2	$A_1 \llbracket \{\} \mid \Sigma \mid \{\} \rrbracket A_2$
d ‘@’, A	$d \bullet A$
‘(’, d, ‘@’, A, ‘)’, ‘(’, E, ‘)’	$(d \bullet A)(E)$
A ‘(’, E, ‘)’	$A(E)$

Table 4: Mapping between CML and *Circus* action CSP-based operators

CML Action	*Circus* Action (ρ)
N, ‘`:=`’, E	$N := E$
‘`atomic`’, ‘`(`’, N1 := E1, ‘`;`’, N2 := E2, ‘`)`’	$N_1, N_2 := E_1, E_2$
‘`[`’,[‘`frame`’ f], [‘`pre`’, E1], ‘`post`’, E2, ‘`]`’,	$\delta(f) : [E_1, E_2]$
‘`if`’ E1, ‘`->`’, A1, ‘`|`’, E2, ‘`->`’, A2, ‘`end`’	$\mathbf{if}\ E_1 \rightarrow A_1 [\!] E_2 \rightarrow A_2\ \mathbf{fi}$
‘`do`’ E1, ‘`->`’, A1, ‘`|`’, E2, ‘`->`’, A2, ‘`end`’	$\mu\, X \bullet$ $\mathbf{if}\ E_1 \rightarrow A_1 ; X$ $[\!] E_2 \rightarrow A_2 ; X$ $[\!] \neg\ (E_1 \vee E_2) \rightarrow \mathit{Skip}$ $\mathbf{fi}$
‘`if`’, E1, ‘`then`’, A1, ‘`elseif`’, E2, ‘`then`’, A2	$\mathbf{if}\ E_1 \rightarrow A_1$ $[\!] E_2 \rightarrow A_2$ $[\!] \neg\ (E_1 \vee E_2) \rightarrow \mathit{Skip}$ $\mathbf{fi}$
‘`if`’, E1, ‘`then`’, A1, ‘`elseif`’, E2, ‘`then`’, A2, ‘`else`’, A3	$\mathbf{if}\ E_1 \rightarrow A_1$ $[\!] E_2 \rightarrow A_2$ $[\!] \neg\ (E_1 \vee E_2) \rightarrow A_3$ $\mathbf{fi}$
‘`cases`’, E, ‘`:`’, p1, ‘`->`’, A1, ‘`,`’ p2, ‘`->`’, A2, ‘`others`’, ‘`->`’, A3, ‘`end`’	$\mathbf{if} \gamma(E, p_1) \rightarrow A_1 [\!] \gamma(E, p_2) \rightarrow A_2$ $[\!] \neg\ (\gamma(E, p_1) \vee \gamma(E, p_2)) \rightarrow A_3$ $\mathbf{fi}$

Table 5: Mapping between CML and *Circus* action command operators

translation of deterministic conditionals, we consider those with one elseif clause. The generalisation of the translation approach for any number of guards and elseif clauses is straightforward. For the nondeterministic loop, we use recursion in the usual way.

For translation of case statements, we use a function γ that takes an expression E and a pattern p as parameters. It defines a condition that captures whether E takes the form described by p. For a pattern $(\_, \_)$, for instance, we get a condition $\exists f_1 : T_1, f_2 : T_2 \bullet E = (f_1, f_2)$. To define γ, by induction on the structure of the pattern, we need information about the type of E.

It is possible to optimise the translation described by ρ by a pre-processing of the model. For example, the several forms of parallelism can be defined in terms of the generalised parallel. This is the case for both parallelism of processes and parallelism of actions. Similarly, the several forms of loop can all be written using the nondeterministic loop construct. We, however, give a more direct translation for the individual constructs.

5.4.2 Correctness

As already explained, the semantics of CML, just like that of ***Circus***, is given in the UTP. In [BBC+12], a semantics is given to the timed version of CML, in terms of a UTP theory. That theory is very similar to the UTP ***Circus*** theory, but its traces are enriched with information about time and incorporates the information about refusals. To establish the correctness of the translation presented above, what we need is to establish the relationship between the timed semantic model of CML and the untimed theory of ***Circus***.

Such work has already been carried out for a different, but closely related, UTP timed theory: that of ***CircusTime***, the timed extension of ***Circus*** [SCHS10]. In the spirit of the UTP, the relationship is established in the form of a Galois connection: a pair of functions that associate establish a correspondence between the elements of the different theories [SCHS10]. These functions do not define a bijection, because the timed model embeds more information.

CML theory We recall that the CML UTP theory has, besides the programming variables, *okay*, and *wait*, a single observation variable *rt* (and their dashed counterparts *okay*$'$, *wait*$'$, and *rt*$'$) [BBC+12]. This is a timed trace: an element of the set defined as follows.

$$timedTrace \mathrel{\widehat=} (\Sigma + \mathbb{P}(\Sigma).tock)^*$$

CML Action	*Circus* Action (ρ)
‘`for`’, N, ‘`in`’, S, ‘`do`’, A	$(\mu X \bullet (seq : \mathrm{seq}\ T \bullet$ $\mathbf{if}\, seq = \langle\rangle \rightarrow Skip$ $[\!]\, seq \neq \langle\rangle \rightarrow N := \mathrm{head}\ seq;$ $A;$ $X(\mathrm{tail}\ seq)$ $\mathbf{fi})$ $)(S)$
‘`for`’, E, ‘`in`’, [‘`reverse`’], s, ‘`do`’, A	$(\mu X \bullet (seq : \mathrm{seq}\ T \bullet$ $\mathbf{if}\, seq = \langle\rangle \rightarrow Skip$ $[\!]\, seq \neq \langle\rangle \rightarrow N := \mathrm{head}\ seq;$ $A;$ $X(\mathrm{tail}\ seq)$ $\mathbf{fi})$ $)(\mathrm{reverse}\ S)$
‘`for`’, ‘`all`’, N, ‘`in set`’, S, ‘`do`’, A	$(\mu X \bullet (set : \mathbb{P}\ T \bullet$ $\mathbf{if}\, set = \emptyset \rightarrow Skip$ $[\!]\, set \neq \emptyset \rightarrow N : [N' \in set];$ $A;$ $X(set \setminus \{N\})$ $\mathbf{fi})$ $)(\mathrm{reverse}\ S)$
‘`for`’, N, ‘`=`’, E1, ‘`to`’, E2, [‘`by`’, E3], ‘`do`’, A	$N := E_1; \mu X \bullet$ $\mathbf{if}\ N \leq E2 \rightarrow A; N := N + E3; X$ $[\!]\, N > E2 \rightarrow Skip$ $\mathbf{fi}$
‘`while`’, E , ‘`do`’, A	$\mu X \bullet \mathbf{if}\ E \rightarrow A; X [\!] \neg\ E \rightarrow Skip\ \mathbf{fi}$

Table 6: Mapping between CML and *Circus* action loop operators

A timed trace uses a special event *tock* to mark the end of each time unit. Just before the *tock*, a set of events records the refusals at the end of that time unit. The trace before the first *tock* or since the previous *tock* identifies the sequence of events that happened in that time unit.

Galois connection As discussed at length in [SCHS10], and hinted in the paper [BBC+12], translation from the timed model to the untimed model can be defined as follows.

Definition 5.1

$$\begin{array}{l} L(P) \mathrel{\widehat{=}} \exists\, rt, rt', tt, tt' : \mathit{timedTrace} \bullet \\ \quad tr = \mathit{traces}_{tt}(rt) \land \mathit{ref} = \mathit{refusals}_{tt}(rt) \\ \quad \land\, tr' = \mathit{traces}_{tt}(rt') \land \mathit{ref}' = \mathit{refusals}_{tt}(rt') \end{array}$$

The function L maps a predicate P of the CML timed theory to a predicate in the untimed ***Circus*** theory. This is achieved by hiding rt and rt' as well as the derived variables tt and tt', while introducing the untimed observation variables: tr, tr', ref, and ref' are obtained by applying the *trace* and $\mathit{refusals}_{tt}$ projection functions to the timed traces. The programming variables, *okay*, and *wait*, and their dashed counterparts, are not affected by L. It establishes a very direct correspondence between the predicates.

The definition of traces_{tt} is simple. It is just a projection that keeps all elements of the trace that are events.

$$\mathit{traces}_{tt}(tt) = tt \upharpoonright \Sigma$$

For $\mathit{refusals}_{tt}$, we consider that trace that has only the refusal sets: all values that are subsets of Σ, and take the last of them.

$$\mathit{refusals}_{tt}(tt) = \mathrm{last}\ (tt \upharpoonright (\mathbb{P}\,\Sigma))$$

The function L is an abstraction; $L(P)$ hides time information and gives a weaker representation of P in the untimed theory. As a result, $L(P)$ can only give a best approximation of the meaning of P, and there might not be an exact inverse of L. It is possible, however, to find a function R which as far as possible undoes the effect of L. Given an untimed predicate Q, R gives the weakest timed predicate with the same behaviour.

Definition 5.2

$$R(Q) \mathrel{\widehat{=}} \sqcap \{P \mid L(P) \sqsupseteq Q\}$$

Because R is a weak inverse of L, then there is an unavoidable loss of information when applying R to the result of an application of L. The following theorem captures this fact based on the refinement order.

Theorem 5.3 $P \sqsupseteq R(L(P))$

Proof:

$R(L(P))$ [definition of R]

$= \sqcap \{Q \mid L(Q) \sqsupseteq L(P)\}$

[L is monotonic, and property of greatest lower bound]

$\sqsupseteq \sqcap \{Q \mid Q \sqsupseteq P\}$ [property of greatest lower bound]

$= P$

If we apply the weakening function R to a predicate Q and then apply the strengthening function L to the result, this may yield a predicate stronger than Q in the untimed theory, as established below.

Theorem 5.4 $L(R(Q)) \sqsupseteq Q$

Proof:

$L(R(Q))$ [definition of R]

$= L(\sqcap \{P \mid L(P) \sqsupseteq Q\}$ [L is monotonic]

$\sqsupseteq \sqcap \{L(P) \mid L(P) \sqsupseteq Q\}$ [property of greatest lower bound]

$\sqsupseteq Q$

These results mean that the functions L and R form a Galois connection. This property guarantees that the timed theory of CML preserves the untimed semantics of programs defined in the *Circus* theory.

As indicated in [SCHS10], the abstraction function L above, when applied to a wait statement, which is available in CML, gives a nondeterministic choice between Skip and Stop. This actually introduces a deadlock state into the program, and therefore liveness properties cannot be explored after the application of this abstraction function. The program that results from the application of L may deadlock, even when the original program does not.

For our purposes, this is not a problem, since we do not relate timed CML models to ***Circus*** models. As already explained, we consider just an untimed subset of CML. This is an issue when, like in [SCHS10], we are interested in analysing the simpler untimed model as a way of obtaining results about the timed model. This kind of technique is of interest in the context of CML, as well as of ***Circus***, but is not in the scope of COMPASS.

Soundness of the translations in Tables 3, 4, 5, and 6 requires us showing that $L(\mathrm{P}) = \rho(P)$, using the CML semantics of P and the ***Circus*** semantics of $\rho(P)$. For the action subset in [BBC^{+}12], the proofs are similar to those in [SCHS10].

As a consequence of these results, we now have two alternatives for applying the systematic approach to build trustworthy CML SoS presented in this document:

1. Translate the CML processes into CSP processes and apply the strategy at the CSP level using CSP tools like FDR, or;
2. Apply the strategy directly at the CML level using the CML model-checker to discharge the side conditions of the rules. This is due to the fact that, besides proving the correspondence between CML and CSP constructs, we have also demonstrated the correspondence between CML and CSP refinement relations for the subset of CML considered here.

6 Case Study

In this section, we describe the specification of a bounded, reactive, buffer as a means of introducing the systematic development approach proposed in this document. This specification is strongly based on that presented in [CSW03] using *Circus*. First, in Section 6.1 we present the basic CML processes. They are then used in Section 6.2 to define the basic contracts, which are systematically composed in order to generate the overall buffer.

The main purpose of this case study is to provide a didactic account of our approach in the context of CML. It is, however, not a realistic example in the context of SoS. As we discuss in Section 7, the application of the strategy to the case studies of COMPASS are planned for Deliverable D24.4 (due in Month 36).

6.1 CML Ring Buffer

In [CSW03], the development of a reactive bounded buffer using the *Circus* refinement calculus resulted in a decentralised buffer that is composed by a ring of cells with a central controller and a cached head. Each single storage cell has its own identification and is able to store one value. The controller is responsible to receive inputs and outputs request from the environment and interacting accordingly with the ring of storage cells. For example, in Figure 10 we present the design of a distributed ring buffer of size 4 (hence, three cells and one place in cache) to which the values 2, 9 and 8 have been written in this order.

It is out of the scope of this document to present the whole development of a centralised buffer into a distributed one as presented in [CSW03]. Here, we focus on the final composition of the basic processes.

First, we assume that the values stored in the buffer are natural numbers. Furthermore, storage cells are identified by natural numbers that range from 1 up to the size of the buffer decremented by one. This is due to the use of a cache in the controller as we will describe later in this section. Finally, every communication between the controller and the storage cells has a direction which is either a request (`req`) or an ancknowledgment (`ack`). All the types are declared in the `types` section of the CML specification.

```
types
    Value = nat
```

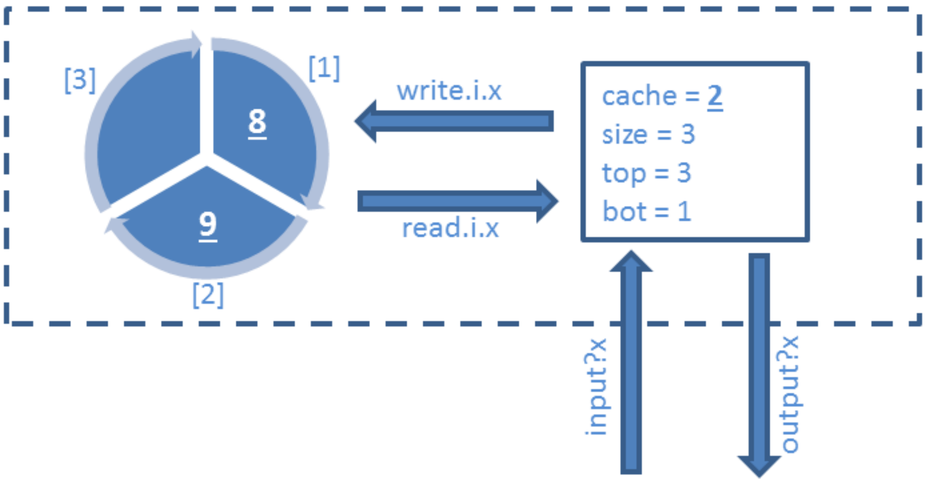


Figure 10: Design of a Distributed Ring Buffer

```
    CellId = nat inv id == id > 0 and id <= maxring
    Direction =  <req> | <ack>
```

The next section of our CML specification of the distributed buffer defines two constants: the size of the buffer, `maxbuff` (for illustration purposes 4), and the number of storage cells in the ring `maxring`.

```
values
    maxbuff = 4;
    maxring = maxbuff - 1
```

The `channels` section specifies the channels used in the specification. The environment is able to interact with the buffer using channels `input` and `output` that carry the value to be stores and retrieved, respectively.

```
channels
    input, output : Value
```

The channels `write` and `read` are used by the controller to exchange values with the ring cells.

```
    write, read: CellId * Direction * Value
```

On the other hand, each individual cell is unaware of the existence of other cells. Hence, from the perspective of the cells, the interaction is via channels `wrt` and `rrd`.

```
    rrd, wrt: Direction * Value
```

We also need to introduce channels that communicate the position of the cell in the ring, as well as the value of the cell, as *rd* and *wrt* do. These channels are used later to make instances of the contract that encapsulates the process `RingCell`; each instance represents an individual storage cell.

```
    rd_i, wrt_i: CellId * Direction * Value
```

We are now able to specify the two basic processes which are used later to define the contracts of our example.

The ring cell is implemented as the following CML process, which has a single state component, `v`, the value stored (if any).

```
process RingCell =
begin
    state v:Value
```

This process has a single operation `setV(x:Value)` that is used to set its component `v` to the value received as argument.

```
    operations
        setV(x:Value)
            frame wr v
            post v = x
```

The ring cell has a single behavioural action, `Act`, in which the cell receives a request to store a value through channel `wrt`, sets its value to the received value using its only operation, acknowledges the writing and then is ready to be read. After receiving a request for a reading, it sends the stored value through an acknowledgment in channel `rrd`.

```
    actions
        Act = wrt.req?x -> setV(x); wrt.ack.x -> Act
              []
              rrd.req?dumb -> rrd.ack!v -> Act
```

This action defines the main behaviour of the storage cell.

```
    @ Act
end
```

This concludes the specification of the storage cell. We now turn to the last basic process, which has a more elaborate specification, the `Controller`.

The controller has four state components: the `cache` that stores the head of the buffer, when the buffer is non-empty; the `size` of the list stored in the buffer; and two indices `bottom` and `top`, to delimit the relevant values.

Modulo arithmetic is used to increment `bot` and `top`. The constant `maxring`, defined as `maxbuff - 1`, gives the bound for the ring.

```
process Controller =
begin
    state cache:Value;
          size:nat;
          top:CellId;
          bot:CellId
```

The initialization operation receives the initial values of the four components and initialises the components accordingly.

```
    operations
        Init(c:Value, s:nat, t:CellId, b:CellId)
            post cache=c and size=s and top=t and bot=b
```

Furthermore, one operation for each state components is provided for setting their values.

```
        SetCache(x:Value)
            frame wr cache:Value
            post cache = x

        SetSize(x:nat)
            frame wr size:nat
            post size = x

        SetTop(x:CellId)
            frame wr top:CellId
            post top = x

        SetBot(x:CellId)
            frame wr bot:CellID
            post bot = x
```

We now describe the `Controller`'s behavioural actions. If the buffer has not reached its maximum size, the action `Input` gets the new input. In the case the buffer is empty, an input is cached. The ring indices do not change and the buffer now contains a single item. If the buffer is not empty, the `Controller` sends the input value to the ring along with the position `top` in which the input is to be stored. This communication is through the channel `write`. In this case, the `cache` is not changed, but the indices and the size of the ring are updated.

```
    actions
        Input =
            [size < maxbuff] &
                input?x ->
                    ( [size = 0] & SetCache(x); SetSize(1)
                      []
                      [size > 0] &
                            write.top.req!x ->
                            write.top.ack?dumb ->
                            SetSize(size+1);
                            SetTop((top mod maxring)+1) )
```

Concerning output, which is only enabled if the buffer is not empty, the value in the *cache* is always the one which is communicated. If the buffer has a single element, communicating this element and updating the `size` are the only relevant actions. Nevertheless, if there are elements stored in any storage cell, the value `x` at position `bot` must be recovered. In this case, the `cache` is updated with this value and `bot` is incremented. The following action captures the necessary case analysis for output. The channel `read` is used to recover the element `x` at position `bot` in the ring.

```
        Output =
            [size > 0] &
                output!cache ->
                    ( [size > 1] &
                        (|~| dumb:Value @
                            read.bot.req.dumb ->
                            read.bot.ack?x -> SetCache(x));
                        SetSize(size-1);
                        SetBot((bot mod maxring)+1)
                      []
                      [size = 1] &
                        SetSize(0))
```

The behaviour of the controller is as follows.

```
    @ Init(0,0,1,1); mu X @ ((Input [] Output); X)
end
```

After initialisation for an empty buffer, inputs and outputs are offered repeatedly, whenever possible.

The ring buffer example, while being appropriate to illustrate the compositional approach described here, is not a realistic example in the context

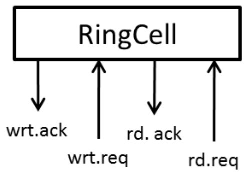


Figure 11: *RingCell* Contract

of SoS: typically, the controller, as a constituent system, would not block because the ring buffer, which would be deployed as another constituent system, is full (recall that the `Input` action will block inputs if the buffer is full). Instead, it would always be allowed to send an input request, and responses (success/failures) would be sent from the constituent system handling the buffer to the constituent system trying the input. Again, our main purpose with this case study is to illustrate the use of the approach in CML. More complex and SoS related case studies are planned for Deliverable D24.4 (due in Month 36).

6.2 BRIC Ring Buffer

Based on both basic processes presented above, we are able to systematically build a process network based on the systematic approach presented in Section 3. First, we define the contracts that encapsulate both processes, which constitute the building blocks of our systematic development approach. As explained in Section 3, a component contract encapsulates a component: it is defined in terms of the component's behaviour (represented as a CML process), ports (represented as channels) and respective interfaces (types).

The contract that encapsulates the `RingCell`, depicted in Figure 11 is defined below.

$$Ctr_{RingCell} \mathrel{\widehat{=}} \left\langle \begin{array}{l} \mathtt{RingCell}, \\ \left\{ \begin{array}{l} \mathtt{rd} \mapsto \mathtt{Direction} \times \mathtt{Value}, \\ \mathtt{wrt} \mapsto \mathtt{Direction} \times \mathtt{Value} \end{array} \right\}, \\ \{\mathtt{Direction} \times \mathtt{Value}\}, \\ \{\mathtt{rd}, \mathtt{wrt}\} \end{array} \right\rangle$$

The contract behaviour is that of the CML process `RingCell`. This component communicates via two channels `rd` and `wrt`, whose types are determined by the second component of the contract.

As explained in Section 3, our model has a higher-level granularity by complementing the syntactical information of a component with behaviour. In our case, we explicitly separated inputs and outputs. The contract $Ctr_{RingCell}$ has the requests as inputs and the acknowledgments as outputs.

$$inputs(\texttt{rd}, Ctr_{RingCell}) = \texttt{\{| rd.req |\}}$$
$$inputs(\texttt{wrt}, Ctr_{RingCell}) = \texttt{\{| wrt.req |\}}$$

$$outputs(\texttt{rd}, Ctr_{RingCell}) = \texttt{\{| rd.ack |\}}$$
$$outputs(\texttt{wrt}, Ctr_{RingCell}) = \texttt{\{| wrt.ack |\}}$$

Besides the restrictions on the $\mathcal{R}$, $\mathcal{I}$, $\mathcal{C}$, which are satisfied, in order for the $Ctr_{RingCell}$ to be a valid component contract, the `RingCell` needs to be an I/O process. This depends on 5 conditions, which are:

1. whenever $\texttt{c.x} \in \alpha(\texttt{RingCell})$, then `c` is an I/O channel;
   - This is correct, since the definitions of the functions *inputs* and *outputs* for channels `rd` and `wrt` above makes it clear that, for both channels, their results are within the productions of the channels on `req` and `ack`, separately, and their intersection is empty.
2. `RingCell` has infinite traces;
   - This is correct because the process has an infinite recursion
3. `RingCell` is divergent-free;
   - This is correct because the process has no hiding.
4. `RingCell` is input deterministic;
   - This is correct because the process does not have internal choice among input events.
5. `RingCell` is strong output decisive.
   - This is correct because there is no choice on outputs; when offered, there is only a single option on outputs.

Hence, we may conclude that $Ctr_{RingCell}$ is a valid contract.

The second contract encapsulates the `Controller` and is depicted in Fig-

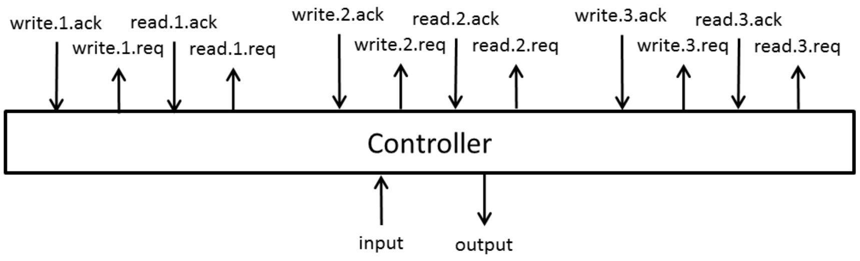


Figure 12: *Controller* Contract

ure 12 is defined below.

$$Ctr_{Controller} \mathrel{\widehat=} \left\langle \begin{array}{l} \texttt{Controller}, \\ \left\{ \begin{array}{l} \texttt{input} \mapsto \texttt{Value}, \texttt{output} \mapsto \texttt{Value}, \\ \texttt{read} \mapsto \texttt{CellId} \times \texttt{Direction} \times \texttt{Value}, \\ \texttt{write} \mapsto \texttt{CellId} \times \texttt{Direction} \times \texttt{Value} \end{array} \right\}, \\ \{\texttt{Value}, \texttt{CellId} \times \texttt{Direction} \times \texttt{Value}\}, \\ \{\texttt{input}, \texttt{output}, \texttt{read}, \texttt{write}\} \end{array} \right\rangle$$

This contract's behaviour is that of the CML process `Controller`, which communicates via channels `input`, `output`, `read` and `write`, whose types are determined by the second component of the contract.

As for the $Ctr_{RingCell}$ contract, we also separated inputs and outputs of the `Controller`'s contract. First, this component inputs on channel `input` and outputs on channel `output`. The reading and writing of values is complementary to the `RingCell`: it has the requests as outputs and acknowledgments as inputs.

$$\begin{array}{l} inputs(\texttt{input}, Ctr_{Controller}) = \texttt{\{| input |\}} \\ inputs(\texttt{output}, Ctr_{Controller}) = \texttt{\{| |\}} \\ inputs(\texttt{read}, Ctr_{Controller}) = \bigcup_{i:CellId} \texttt{\{| read.i.ack |\}} \\ inputs(\texttt{write}, Ctr_{Controller}) = \bigcup_{i:CellId} \texttt{\{| write.i.ack |\}} \end{array}$$

$$\begin{array}{l} outputs(\texttt{input}, Ctr_{Controller}) = \texttt{\{| |\}} \\ outputs(\texttt{output}, Ctr_{Controller}) = \texttt{\{| output |\}} \\ outputs(\texttt{read}, Ctr_{Controller}) = \bigcup_{i:CellId} \texttt{\{| read.i.req |\}} \\ outputs(\texttt{write}, Ctr_{Controller}) = \bigcup_{i:CellId} \texttt{\{| write.i.req |\}} \end{array}$$

Besides the restrictions on the $\mathcal{R}$, $\mathcal{I}$, $\mathcal{C}$, which are satisfied, in order to be a valid component contract, the `Controller` needs to be an I/O process. This depends on the same conditions as those for the `RingCell` presented above.

1. whenever $\texttt{c.x} \in \alpha(\texttt{Controller})$, then `c` is an I/O channel;
   - This is correct, since the definitions of the functions *inputs* and *outputs* presented above clearly have an empty intersection.
2. `Controller` has infinite traces;
   - This is correct because the process has an infinite recursion
3. `Controller` is divergent-free;
   - This is correct because the process has no hiding.
4. `Controller` is input deterministic;
   - This is correct because the process does not have internal choice among input events.
5. `Controller` is strong output decisive.
   - This is only correct because the `dumb` value communicated in `read.bot.req.dumb` is non-deterministically chosen. Otherwise, if an external choice on such value or an input `read.bot.req?dumb` were offered, the process would not satisfy this property.

Hence, we may conclude that $Ctr_{RingCell}$ is a valid contract.

We are now able to systematically compose the basic contracts in order to generate the overall buffer.

The contracts related to each individual storage cell are defined as instantiations of the contract $Ctr_{RingCell}$.

$$\begin{aligned}
Cell_1 &\mathrel{\widehat=} COMP_{Inst}(Ctr_{RingCell}, \{\texttt{rd} \mapsto \texttt{rd\_i.1}, \texttt{wrt} \mapsto \texttt{wrt\_i.1}\})\\
Cell_2 &\mathrel{\widehat=} COMP_{Inst}(Ctr_{RingCell}, \{\texttt{rd} \mapsto \texttt{rd\_i.2}, \texttt{wrt} \mapsto \texttt{wrt\_i.2}\})\\
Cell_3 &\mathrel{\widehat=} COMP_{Inst}(Ctr_{RingCell}, \{\texttt{rd} \mapsto \texttt{rd\_i.3}, \texttt{wrt} \mapsto \texttt{wrt\_i.3}\})
\end{aligned}$$

Each instance renames the channels for reading and writing in order to identify the cells in the communication accordingly. We are left with the component contracts presented in Figure 13. For the same reasons as those described for the generic ring cell above, each of the contract instantiation above is indeed a valid component contract.

6.2.1 BRIC Composition

The next step is to verify the application of the composition rules that can be used to compose the distributed ring. The first composition is an interleave

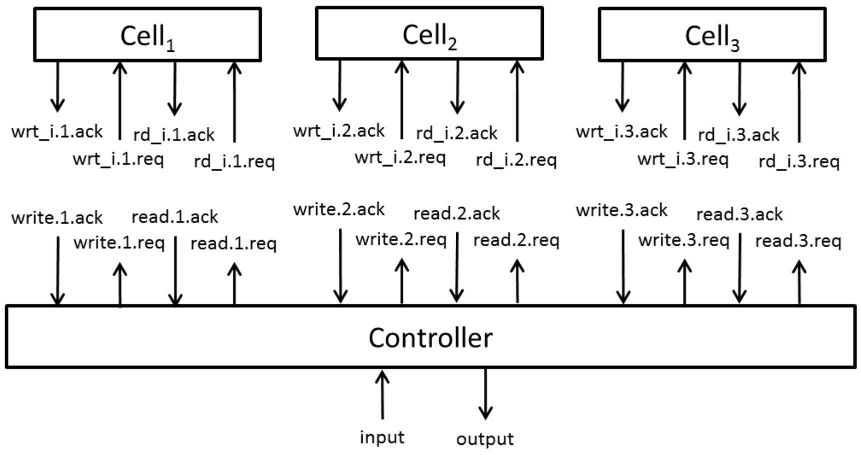


Figure 13: Contracts Before Composition

between cells 1 and 2.

$$DRing_{1\_2} \mathrel{\widehat=} Cell_1 \; [|||] \; Cell_2$$

Since their alphabets are different (see the instantiation above) the rule application is valid.

The result of this composition is the contract $DRing_{1\_2}$. Our systematic development approach proposed in this document guarantees the deadlock-freedom of the resulting contract based on the deadlock-free of the composing contracts. The same applies to the remaining compositions in this section.

For the same reasons as the first composition, the interleave composition of the resulting process with the last cell is also valid.

$$DRing \mathrel{\widehat=} DRing_{1\_2} \; [|||] \; Cell_3$$

We are left with the structure presented in Figure 14, in which we have two independent component contracts: one that represents the storage cells, *DRing* and the one the represented the `Controller`.

Next, we use the communication rule to link both contracts together on events $write.1$ and $wrt\_i.1$.

$$CRingCell_{1\_wrt} \mathrel{\widehat=} Ctr_{Controller}[write.1 \leftrightarrow wrt\_i.1]DRing$$

This composition is only valid if the channels are in the corresponding process' alphabets and these alphabets do not intersect; both conditions are

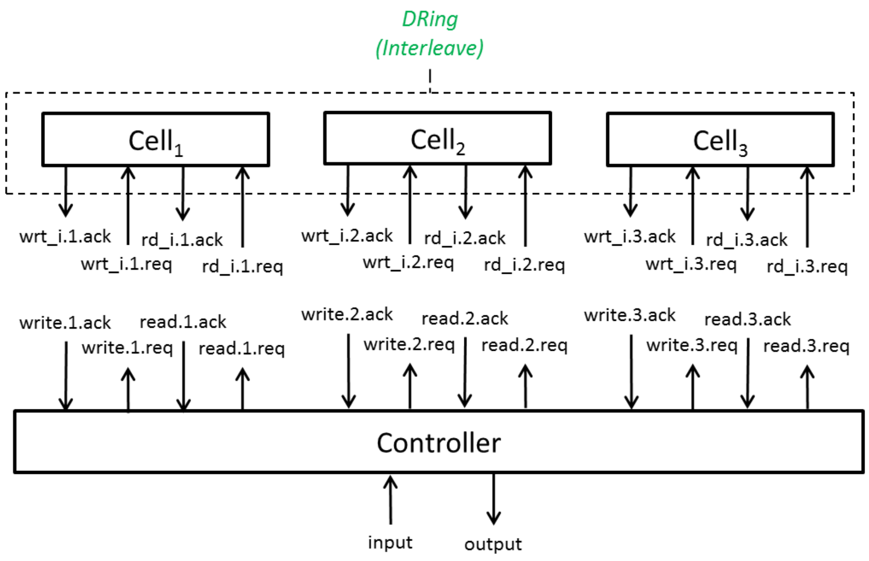


Figure 14: Contracts Before Communication

satisfied. The next conditions are related with the port protocols, which are defined in the sequel.

The protocol implementation of the cells on channel $wrt\_i.1$, as expected, is deterministic on the inputs and non-deterministic on the outputs. It is important to notice that the protocol has control on the values that are read and written (to and from the cell). It only enforces req before ack.

$$\begin{aligned}
&Prot_{IMP}(DRing, wrt\_i.1)\\
&\quad = DRing \upharpoonright \{\!|\ wrt\_i.1\ |\!\}\\
&\quad = \sqcap\, v2 : Value \bullet\\
&\qquad\quad wrt\_i.1.req?v1 \rightarrow wrt\_i.1.ack.v2 \rightarrow Prot_{IMP}(DRing, wrt\_i.1)
\end{aligned}$$

Furthermore, we also define the corresponding sets on $inputs$ and $outputs$.

$$\begin{aligned}
&inputs(Prot_{IMP}(DRing, wrt\_i.1), wrt\_i.1) = \{\!|\ wrt\_i.1.req\ |\!\}\\
&inputs(Prot_{IMP}(DRing, rd\_i.1), rd\_i.1) = \{\!|\ rd\_i.1.req\ |\!\}\\
&outputs(Prot_{IMP}(DRing, wrt\_i.1), wrt\_i.1) = \{\!|\ wrt\_i.1.ack\ |\!\}\\
&outputs(Prot_{IMP}(DRing, rd\_i.1), rd\_i.1) = \{\!|\ rd\_i.1.ack\ |\!\}
\end{aligned}$$

Using the definition for protocol implementation presented in Section 4, we have that the protocol is valid if, and only if:

1. $Prot_{IMP}(DRing, wrt\_i.1)$ is an I/O Process, that is, as explained before.

   (a) whenever $c.x \in \alpha Prot_{IMP}(DRing, wrt\_i.1)$, then c is an I/O channel;
   (b) $Prot_{IMP}(DRing, wrt\_i.1)$ has infinite traces;
   (c) $Prot_{IMP}(DRing, wrt\_i.1)$ is divergent-free;
   (d) $Prot_{IMP}(DRing, wrt\_i.1)$ is input deterministic;
   (e) $Prot_{IMP}(DRing, wrt\_i.1)$ is strong output decisive.
2. $inputs(Prot_{IMP}(DRing, wrt\_i.1), wrt\_i.1) \subseteq inputs(DRing, wrt\_i.1)$
3. $outputs(Prot_{IMP}(DRing, wrt\_i.1), wrt\_i.1) \subseteq outputs(DRing, wrt\_i.1)$
4. $\alpha(Prot_{IMP}(DRing, wrt\_i.1)) \subseteq \{\| wrt\_i.1 \|\}$
5. $DRing$
   $\equiv_{\mathrm{T}}$
   $DRing \; [\![\Sigma]\!] \; (Prot_{IMP}(DRing, wrt\_i.1) \; ||| \; RUN(\Sigma \setminus \{\| wrt\_i.1 \|\}))$

Next, we need to apply the rename (R_{IO}) to the protocol as follows:

$$
\begin{aligned}
& Prot_{IMP}(DRing, wrt\_i.1) \; [\![R_{IO}^{wrt\_i.1 \mapsto write.1}]\!] \\
& = \sqcap v2 : Value \bullet \\
& \qquad wrt\_i.1.req?v1 \rightarrow wrt\_i.1.ack.v2 \rightarrow \\
& \qquad Prot_{IMP}(DRing, wrt\_i.1) \; [\![R_{IO}^{wrt\_i.1 \mapsto write.1}]\!] \\
& = \mu X \bullet \sqcap v2 : Value \bullet \\
& \qquad wrt\_i.1.req?v1 \rightarrow wrt\_i.1.ack.v2 \rightarrow X \; [\![R_{IO}^{wrt\_i.1 \mapsto write.1}]\!] \\
& = \mu X \bullet \sqcap v2 : Value \bullet \\
& \qquad wrt\_i.1.req?v1 \rightarrow write.1.ack.v2 \rightarrow X
\end{aligned}
$$

Let us call this process, PR_DRing.

The next condition for the composition is that PR_DRing must satisfy the Finite Output Property (FOP). That is true if, and only if, $PRRing \setminus (outputs(PRRing))$ is divergence-free. By definition of *inputs* and *outputs* of processes to which R_{IO} has been applied, we have:

$$\begin{array}{l}
\mathit{inputs}(\mathit{Prot}_{\mathit{IMP}}(\mathit{DRing}, \mathit{wrt\_i}.1) \llbracket R_{IO}^{\mathit{wrt\_i}.1 \mapsto \mathit{write}.1} \rrbracket) \\
= \mathit{inputs}(\mathit{Prot}_{\mathit{IMP}}(\mathit{DRing}, \mathit{wrt\_i}.1 \mapsto \mathit{write}.1)) \\
= \{\!|\ \mathit{wrt\_i}.1.\mathit{req}\ |\!\} \\
\text{and} \\
\mathit{outputs}(\mathit{Prot}_{\mathit{IMP}}(\mathit{DRing}, \mathit{wrt\_i}.1) \llbracket R_{IO}^{\mathit{wrt\_i}.1 \mapsto \mathit{write}.1} \rrbracket) \\
= \mathit{outputs}(\mathit{Prot}_{\mathit{IMP}}(\mathit{DRing}, \mathit{wrt\_i}.1))\,[\mathit{wrt\_i}.1 \leftarrow \mathit{write}.1] \\
= \{\!|\ \mathit{wrt\_i}.1.\mathit{ack}\ |\!\}\,[\mathit{wrt\_i}.1 \leftarrow \mathit{write}.1] \\
= \{\!|\ \mathit{write}.1.\mathit{ack}\ |\!\}
\end{array}$$

That means, it satisfies FOP, if, and only if, $\mathit{PR\_DRing} \setminus \{\!|\ \mathit{write}.1.\mathit{ack}\ |\!\}$ is divergence-free, which is clearly true.

At this point, it is extremely important to highlight the reasons for having separated the communications on *req*uests and *ack*nowledgments. Clearly, from the analysis above, if we did not have this separation, we would not be able to define port protocols with the finite output property.

Finally, the protocol $\mathit{Prot}_{\mathit{IMP}}(\mathit{DRing}, \mathit{wrt\_i}.1)$ is clearly I/O Confluent since there are no alternative choices on different channels. This is automatically checked by a model checker, by checking that

$$\mathit{InBufferProt}(\mathit{Prot}_{\mathit{IMP}}(\mathit{DRing}, \mathit{wrt\_i}.1), \mathit{wrt}_i.1)$$

is deterministic, where *InBufferProt* performs a data forgetful renaming on the given process and then places an input one-place inwards pointing buffer on every individual event of the renamed process. The process *InBufferProt* below represents these steps altogether.

$$\begin{array}{l}
CP(a, b) = a \rightarrow b \rightarrow CP(a, b) \\
C(a, P) = (P[[a \leftarrow \mathit{mid}]] \ \llbracket \{\!|\ \mathit{mid}\ |\!\} \rrbracket \ CP(a, \mathit{mid})) \setminus \{\!|\ \mathit{mid}\ |\!\} \\
\mathit{CIO}(P) = C(\mathit{in}, C(\mathit{out}, P))
\end{array}$$

$$\begin{array}{l}
\mathit{InBufferProt}(P, c) = \\
\qquad \mathit{CIO}(P[[x \leftarrow \mathit{in}, y \leftarrow \mathit{out} \mid x \leftarrow \mathit{inputs}(P), y \leftarrow \mathit{outputs}(P)]])
\end{array}$$

In a very similar manner, we define the protocol implementation of the controller on channel *write*.1.

$$\begin{array}{l}
\mathit{Prot}_{\mathit{IMP}}(\mathit{Ctr}_{\mathit{Controller}}, \mathit{write}.1) \\
\qquad = \mathit{DRing} \upharpoonright \{\!|\ \mathit{wrt\_i}.1\ |\!\} \\
\qquad = \bigsqcap v1 : \mathit{Value} \bullet \\
\qquad\qquad \mathit{wrt\_i}.1.\mathit{req}.v1 \rightarrow \mathit{wrt\_i}.1.\mathit{ack}?v2 \rightarrow \mathit{Prot}_{\mathit{IMP}}(\mathit{DRing}, \mathit{wrt\_i}.1)
\end{array}$$

Its definition is very similar to that of the ring cell; however, since it outputs on the requests and inputs on the acknowledgments, the internal choice is left to the value communicated on the request. For the same reasons as discussed above, which we omit here for conciseness, this protocol is also a valid one, and its renamed version (R_{IO}) satisfies the finite output property. Furthermore, the protocol is also I/O Confluent since there are no alternative choices on different channels.

The final requirement of the rule application is the strong compatibility of the renamed versions of the protocols. Our experiments demonstrated that the protocols are strong compatible, since at every possible trace there is an output from either one of the protocols and the other protocol accepts the existing output as an input. More precisely, we have that:

$$\begin{array}{l}
\mathit{inputs}\ (\mathit{Prot}_{\mathit{IMP}}(\mathit{DRing}, \mathit{wrt\_i}.1) \llbracket R_{IO}^{\mathit{wrt\_i}.1 \mapsto \mathit{write}.1} \rrbracket) = \\
\qquad \{\!|\ \mathit{wrt\_i}.1.\mathit{req}\ |\!\} \\
\mathit{outputs}(\mathit{Prot}_{\mathit{IMP}}(\mathit{DRing}, \mathit{wrt\_i}.1) \llbracket R_{IO}^{\mathit{wrt\_i}.1 \mapsto \mathit{write}.1} \rrbracket) = \\
\qquad \{\!|\ \mathit{write}.1.\mathit{ack}\ |\!\} \\
\mathit{inputs}\ (\mathit{Prot}_{\mathit{IMP}}(\mathit{Ctr}_{\mathit{Controller}}, \mathit{write}.1) \llbracket R_{IO}^{\mathit{write}.1 \mapsto \mathit{wrt\_i}.1} \rrbracket) = \\
\qquad \{\!|\ \mathit{write}.1.\mathit{ack}\ |\!\} \\
\mathit{outputs}(\mathit{Prot}_{\mathit{IMP}}(\mathit{Ctr}_{\mathit{Controller}}, \mathit{write}.1) \llbracket R_{IO}^{\mathit{write}.1 \mapsto \mathit{wrt\_i}.1} \rrbracket) = \\
\qquad \{\!|\ \mathit{wrt\_i}.1.\mathit{req}\ |\!\}
\end{array}$$

Hence, the renamed versions of the protocols are strong compatible because the following conditions are, indeed, satisfied.

- $\mathit{outputs}(\mathit{Prot}_{\mathit{IMP}}(\mathit{DRing}, \mathit{wrt\_i}.1) \llbracket R_{IO}^{\mathit{wrt\_i}.1 \mapsto \mathit{write}.1} \rrbracket)$
  $\subseteq \mathit{inputs}(\mathit{Prot}_{\mathit{IMP}}(\mathit{Ctr}_{\mathit{Controller}}, \mathit{write}.1) \llbracket R_{IO}^{\mathit{write}.1 \mapsto \mathit{wrt\_i}.1} \rrbracket)$
- $\mathit{outputs}(\mathit{Prot}_{\mathit{IMP}}(\mathit{Ctr}_{\mathit{Controller}}, \mathit{write}.1) \llbracket R_{IO}^{\mathit{write}.1 \mapsto \mathit{wrt\_i}.1} \rrbracket)$
  $\subseteq \mathit{inputs}(\mathit{Prot}_{\mathit{IMP}}(\mathit{DRing}, \mathit{wrt\_i}.1) \llbracket R_{IO}^{\mathit{wrt\_i}.1 \mapsto \mathit{write}.1} \rrbracket)$
- $\mathit{outputs}(\mathit{Prot}_{\mathit{IMP}}(\mathit{DRing}, \mathit{wrt\_i}.1) \llbracket R_{IO}^{\mathit{wrt\_i}.1 \mapsto \mathit{write}.1} \rrbracket)$
  $\cap\ \mathit{outputs}(\mathit{Prot}_{\mathit{IMP}}(\mathit{Ctr}_{\mathit{Controller}}, \mathit{write}.1) \llbracket R_{IO}^{\mathit{write}.1 \mapsto \mathit{wrt\_i}.1} \rrbracket) = \emptyset$
- $\mathit{inputs}(\mathit{Prot}_{\mathit{IMP}}(\mathit{DRing}, \mathit{wrt\_i}.1) \llbracket R_{IO}^{\mathit{wrt\_i}.1 \mapsto \mathit{write}.1} \rrbracket)$
  $\cap\ \mathit{inputs}(\mathit{Prot}_{\mathit{IMP}}(\mathit{Ctr}_{\mathit{Controller}}, \mathit{write}.1) \llbracket R_{IO}^{\mathit{write}.1 \mapsto \mathit{wrt\_i}.1} \rrbracket) = \emptyset$

This concludes the verification of the conditions that validate the communication composition of *DRing* and the $\mathit{Ctr}_{\mathit{Controller}}$.

After the communication composition, we are left with a single component depicted in Figure 15. Further compositions require the application of rules that allow the connection of channels of a same component. In our approach, we have two options: feedback and reflection. The former is cheaper regarding

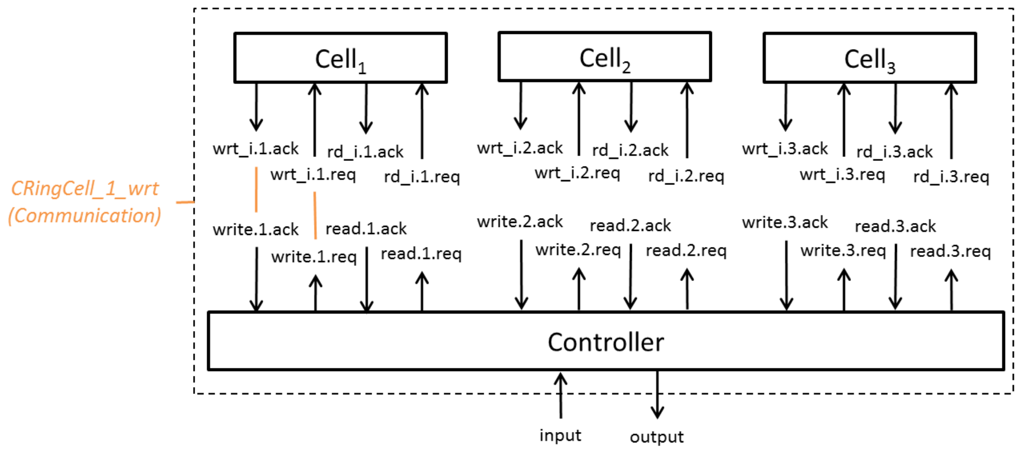


Figure 15: $CRingCell_{1\_wrt}$ Contract

verification costs and for this reasons, we first try to apply it before using the latter.

The first feedback composition connects channels $rd\_i.1$ and $read.1$.

$$CRingCell_1 \mathrel{\widehat{=}} CRingCell_{1\_wrt}[read.1 \hookrightarrow rd\_i.1]$$

The vast majority of the conditions that validate the application of this rule are the extremely similar to those of the last rule application. The only difference is on the channels used. For conciseness, we omit a discussion about these conditions here since it would be almost a repetition of the discussion presented above. The feedback composition, however, has a further condition: the channels that are being connected ought to be decoupled, that is, communication through both channels of a same process behaves as communications between channels of distinct processes (channels are independent). In this case, our experiments demonstrate that the channels are indeed decoupled; hence, the rule application is valid. Intuitively, the channels are decoupled because by connecting $rd\_i.1$ and $read.1$ we have not introduced any cycle of dependence.

Similarly, we might make further use of the feedback composition as long as we do not introduce such cycles of dependence.

$$CRingCell_{2\_wrt} \mathrel{\widehat{=}} CRingCell_1[write.2 \hookrightarrow wrt\_i.2]$$
$$CRingCell_2 \mathrel{\widehat{=}} CRingCell_{2\_wrt}[read.2 \hookrightarrow rd\_i.2]$$

The application of all feedback composition leave us with $CRingCell_2$ depicted in Figure 16. By looking at this figure and bearing in mind the

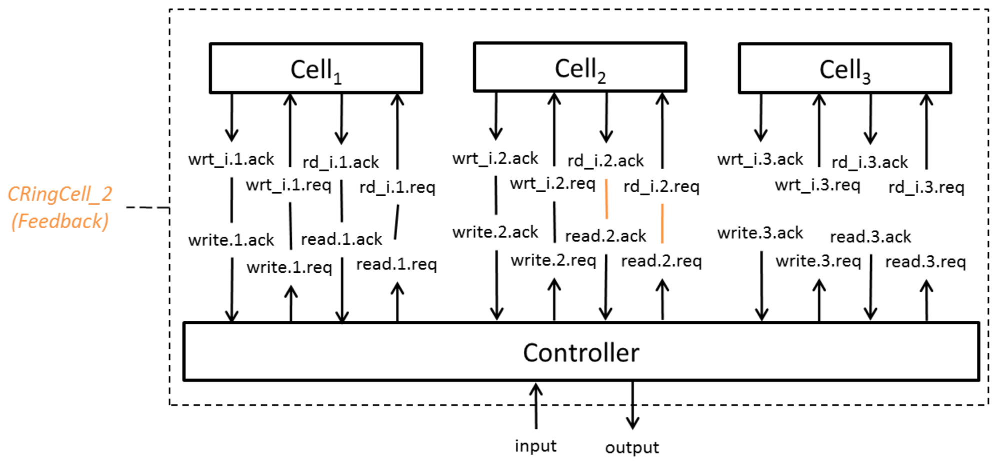


Figure 16: $CRingCell_2$ Contract

behaviour of a circular buffer, we might intuitively conclude that any further communication will not be among decoupled channels because we will be introducing a cycle of behavioural dependence. In fact, our experiments demonstrate that the channels $write.3$ and $wrt\_i.3$ are not decoupled. The same conclusion applies to channels $read.3$ and $rd\_i.3$. For this reason, the conclusion of the systematic development of our case study uses the last composition rule, the reflexive rule.

Our first use of the reflexive rule connects channels $write.3$ and $wrt\_i.3$. Rather than requiring channels to be decoupled like the feedback rule, the reflexive rule requires the projection of the overall process on both channels to be buffering self-injection compatible (which includes being strong compatible) and to satisfy the finite output property. In our work, we use the following lemma, originally presented and proved in [Ram11], which shows that a buffering self injection compatible process can establish a communication between its channels via a one-place buffer without deadlock. Again, the application of this result at the level of CML is only allowed because of the results presented in Section 5.

Lemma 6.1 *Let P be an deadlock-free I/O process, c and z communication channels, and LR_1 and LR_2 bijections, such that:*

1. $LR_1 : outputs(P, c) \leftrightarrow inputs(P, z)$;
2. $LR_2 : outputs(P, z) \leftrightarrow inputs(P, c)$;
3. $Prot_{IMP}(P, c) \parallel[\, LR_1]\parallel$ *and* $Prot_{IMP}(Q, z) \parallel[\, LR_2]\parallel$ *are strong compatible, and;*

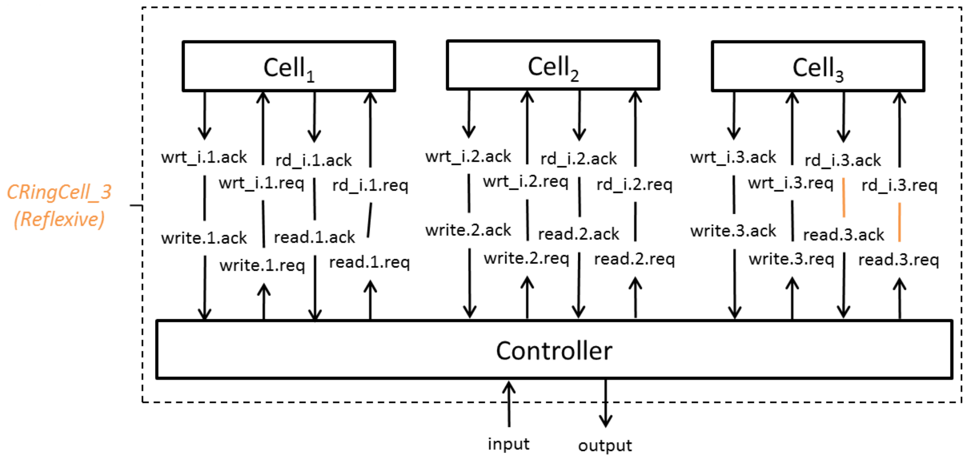


Figure 17: *CRingCell* Contract

4. *$Prot_{IMP}(P, c)$ and $Prot_{IMP}(Q, z)$ satisfy the finite output property.*

Then, $P \upharpoonright \{c, z\}$ *is* buffering self-injection compatible *if, and only if, the following process is deadlock-free:*

$$P \upharpoonright \{c, z\} \;[\![\{c, z\}]\!]\; BUFF^1_{IO}(LR_1, LR_2)$$

The verification of strong compatibility and the finite output property is achieved in the same manner as previously discussed in this section. Finally, the verification that the process can communicate via a one-place buffer without deadlock is achieved simply by model checking. In our experiments, both rule applications below satisfy all these properties; hence, they are valid.

$$CRingCell_{3\_wrt} \mathrel{\widehat=} CRingCell_2[write.3 \hookrightarrow wrt\_i.3]$$
$$DBuffer \mathrel{\widehat=} CRingCell_{3\_wrt}[read.3 \hookrightarrow rd\_i.3]$$

□

It is important to point out that the verification of that the process can communicate via a one-place buffer without deadlock is non-compositional. For this reason, the application of the reflexive rule has a very high cost of verification and is currently our bottleneck. We are developing alternatives for the application of this rule that are not in the context of this document. So far, the alternatives have proved to be extremely efficient.

This concludes the systematic development of a distributed buffer depicted in Figure 17. This buffer interacts only via channels *input* and *output*. This interaction is only on these channels because the composition rules removes the connecting channels from the contracts interfaces. This prunes the possibility of further connections on these channels.

Using a CSP version that corresponds to the CML buffer, we demonstrated in FDR that this CSP BRIC version of the buffer is indeed a refinement of an abstract buffer. The verification of the correctness of the CML BRIC version of the buffer requires the implementation of the CML model checker which is currently under development.

7 Conclusion

Although compositional approaches provide mechanisms and tools for constructing systems by plugging components together, the safe construction of these systems is still a research challenge. Trustworthiness is required during several development activities, such as safe composition of third-party components or the correct adaptation of library components.

In this document, we have proposed a correct-by-construction approach for building trustworthy CML SoS. The approach focuses on performing analyses that are intended to address engineering concerns on compositional development. In special, we focus on component integration. The entire approach is based on the original approach from [Ram11] that is underpinned by the CSP process algebra, which offers rich semantic models that support a wide range of process verifications, and comparisons. The strategy for lifting the entire approach from CSP to CML (via ***Circus***) provides a general theoretical link between these three formal languages that fosters the reuse of practical results achieved in any of them. These results contribute with the development of compositional design and analysis techniques, based on sophisticated architectural patterns (WP24); they will help to realise the potential and promise of SoS.

Another account of the soundness of our compositional analysis technique is being developed in the form of a Hoare Logic for CML. It supports rigorous reasoning about CML programs using rules like those we presented, which are, however, expressed as inference rules proved directly in the CML UTP theory. It is a formal logic system used to prove Hoare Triples: statements of the form $\{P\}C\{Q\}$, asserting that a precondition (P) and postcondition (Q) is applicable to the model C of a CML program (an element of the CML UTP theory). It consists of axioms for SKIP, STOP and assignment, and inference rules prefixing, parallel and sequential composition, internal and external choice, timeout, recursion, and hiding operators. These can be combined to develop proofs for composite programs.

The axioms determine P, in terms of an arbitrary Q, such that $RT(P \vdash Q) \sqsubseteq C$ holds, where RT is the healthiness condition of the CML UTP theory. Given a hypothesis containing Hoare triples for the constituent programs of C, the inference rules give P and Q, in terms of the hypothesis assertions. Side conditions on hypothesis assertions are used to reduce the overall complexity of a rule. The rules are proved sound in the same UTP theory of the programs, that of CML, and also exact in the sense that the precondition is both necessary and sufficient for the command establishing the postcondition. For cases when only sufficiency is needed, simpler rules will also be given as part of our future work.

The CML BRIC component model is aligned to other models with behaviour descriptions. It focuses on (re)active components that are *input deterministic* and *output decisive*. Reuse and compositions are allowed not only to components, but also to connectors. Furthermore, it considers not only compositions between two distinct components, but also the assembly between ports of the same component. This brings more flexibility to design decisions at development. An operation for hiding information to pack components into black-boxes is also presented.

We present a comprehensive set of composition rules that can be regarded as safe steps in the development. The application of the rules can be used to systematically develop a wide variety of trustworthy component systems, and guarantees, by construction, the absence of deadlock. The approach covers not only tree-topologies, but also topologies with cycles in a compositional method, without being aware of needing to know the overall structure of the system. Port protocols play an important role in the approach, and, in conjunction with other properties, help to alleviate verifications by supporting local analyses.

We improve the verification using enriched components with metadata. We propose an integrated correct-by-construction approach for component contracts using metadata, which extends our approach for arbitrary components with improved and lightweight side conditions. Metadata are derived from component-contract elements and are used in substitution to heavier verifications in the version without metadata. Additionally, metadata of compositions can be easily derived from the metadata of its constituting components. As a result, the order of complexity of the verifications is drastically reduced.

Finally, the application of our approach has been illustrated in a case study in Section 6: *the Ring Buffer*.

Despite these contributions, the proposed approach has some limitations:

1. The benefits of using metadata are limited to the application of composition and feedback composition rules. Although this corresponds to two of the four basic proposed composition rules, the application of the other composition rules is compatible with our strategy with metadata. Moreover, one of these composition rules, the interleave one, is already very simple, and does not need further improvements.

2. The strategy with metadata indicates that some compatible communications between components, as incompatible (false-negatives). This is an intrinsic problem in local analysis methods, which is acceptable considering the advantages that it brings in scalability. In these scenarios, the developers have to use traditional verification methods to complement our strategy. Furthermore, the strategy with metadata must be adopted as a technique that guides the attention of the integrator to the most crucial compositions, and not as a 'silver bullet' method for the composition problem in general.

7.1 Related work

The topic of this document expands over fields of architecture and reliability modelling. We have studied a variety of approaches in each domain, and identified a few approaches that span both domains. Here, we first present a summary of related approaches to architectural modelling. We then provide an overview of existing reliability models. The former focuses on the characteristics of our component model, and the latter on the constructive constraints to ensure properties of component-based systems. Since there is an extensive number of works in these fields, we focus on the works being most influential and related to our work.

There are several different approaches to component models. As pointed out in [Wal03], each component model is designed to achieve specific goals. Furthermore, each one has its benefits and deficiencies, depending on the context in which it is analysed. In this section, we consider related works and compare them according to the context of this document. For instance, there are multiple (modelling) aspects for component [RM04].

To begin with, we do not relate our work with other component models that define low-level granularity components, in which contracts/interfaces capture solely syntactical information (like method signatures). Low-level granularity component models are associated to component technologies found in industry that are usually designed to support quick development or to permit the use of different programming languages in development. These are furthermore not designed for reasoning. In order to get around this limitation concerning interface representation, several authors [FLF01, LD00, LW94] propose the specification of the 'behaviour' part via pre- and post conditions and invariants. According to [Pla05], one of the key obstacles in applying these approaches to components is that they require an explicit capturing of (object) state - this may be both a very hard-to-achieve and, potentially, limiting decision at an early stage of a component design.

We focus on works that support behaviour description of entities. The idea of expressing behaviour of an object as a regular process (via traces as sequences of method calls) has been published in [Nie93]. It even considers the role of client calls (in a simple case) via parallel composition. The importance of capturing behaviour of components as sequences of events for COTS components (commercial off the shelf) is emphasized also in [DR02] where a way of identifying behaviour via monitoring experiments is described.

There have been a huge number of publications on behaviour description of components and connectors [ADG98, BCD02, HLL06b, BHP06, Arb04, Sif10, CZ07]. Our approach integrates aspects from different but closely related domains. The target concrete syntax of our work is CSP, but the elements within BRIC component contracts (see Definition 3.1) are not directly represented by this notation. CSP is used to give the underlying semantics of our component model, and to help verifications. However, there are more suitable concrete syntaxes to represent our notions at development phase, such as Architectural Description Languages (ADLs) [MT00] or the modelling languages UML-RT [SR98] and UML2 [Obj07]. The concepts in these languages are highly compatible with our component model, and one can benefit from using both approaches, like modelling in one language and performing verifications in another.

Our component model is based on I/O transition systems, has explicit architectural structure, and presents connectors as first class design elements. These characteristics resemble several ADL approaches, such as Wright [All97b, ADG98], Darwin [MK96], PADL [BCD02], and ROOM [SGW94]. Our component model focuses on design elements, and does not take into consideration the expressiveness of programming languages as architectural programming models, such as ArchJava [ACN02], SOFA [BHP06], Fractal [BCL$^+$06], rCOS [HLL06a, CHLZ07] and BIP [Sif10]; the design concepts in these ADLs are, however, compatible with concepts in our component model. Another related ADL is ROOM [SGW94], which later evolved to UML-RT [SR98], which in the meantime has been incorporated into UML2 [Obj07].

Despite their similarities, the representation of components in these works differs in some extent. Some consider the internal behaviour of components, e.g. [BCL$^+$06, HLL06a], other the external behaviour, e.g. [ADG98]. Some component models represent components solely by their port protocols, e.g. [CZ07], other neglects this kind of behaviour, e.g. [HLL06a]. In our work, we discriminate the external behaviour of components and their points of interactions (port protocols). Component contracts have the whole external component behaviour, or are enriched with port protocols (see component contracts and metadata in Chapter 3). Each kind of behaviour has its benefits in reasoning. Port protocols alleviate verifications, whereas the whole behaviour of components is essential for structural analysis of larger systems. The comparison with approaches to verify component-based systems is presented in the next section. Our component model also has operations to hide information in component contracts. The wrapping operation hides the part of the component behaviour that is not available for composition (the interaction between the sub-components of the composition). This is, however, different from the concept of *publication* presented in rCOS [ZKL10] for creating 'black-box components'. In rCOS, a publication is an abstraction of a contract that removes behavioural information from the contract.

Another important issue is the representation of connectors. Some works have an explicit representation for connectors, e.g. [ADG98], others connectors are not distinct from components, e.g. [HLL06a]. In some approaches both in components and connectors can be reused. Our approach is closest to that of [Spi04], in which, at the design level, connectors are represented as parametrised CSP processes, called *connector wrapper* templates. At the integration phase, connectors have the same representation as components [ADG98, Spi04]. This provides means of enhancing existing connectors at different levels of abstraction, which is aligned with practical approaches of connector representation [MB05]. The more abstract one is used at design and it is meant for reuse. The more concrete one has the same structure of components, and it can, therefore, be used as units of compositions.

This issue is related to coordination languages. In these languages, connectors are used to coordinate component interactions. Compared to ADL connectors, these connectors can represent much more sophisticated coordination policies for sets of components. In the coordination language Reo [Arb04], complex connectors are constructed from a composition of a comprehensive set of basic connectors. The computational aspects of the connectors are therefore limited to these basic connectors. In our work, we do not focus on coordination issues. Apart from that, one can build exogenous coordination on the top of our component model. Connectors can also be built from more basic ones, and, at any level, connectors can have complex behaviours.

Most of these component models [All97b, BCD02, HLL06b, Sif10, CZ07] have an underlying semantics, which allows verifications; most of these component models are classified as ADL, and typically subsumes a formal semantic theory [MT00]. In the next section, we discuss the relations between our approach and the verification methods in these works.

There are several efforts on the verification of Component-based Systems [BCD02, MCM08, HJK10b, MW97, All97b]. The scalability issue in compositional verification has been actively addressed in this field; compositional verification is based on the idea that the correctness check of a complex system can be divided into smaller verification tasks for its components. Here, we compare our work, not only with approaches with an explicit component model, but also with others that focus on the verification of behavioural elements (which may not be fully aligned with a component development method).

The work reported in [GGMC$^+$07, MCMM07, MCM07] presents an extensive study of quality properties in CBS. It discusses liveness, local progress, deadlock, fairness and robustness. We implicitly discuss these properties, except fairness and robustness. The deadlock property is locally addressed by our compatibility notion, which is an important condition of our composition rules. Therefore, deadlock is preserved by our composition rules for BRIC components, and local progress is also preserved when composition rules are applied for BRICK components (components with metadata). None of the composition rules introduce livelock. Relating to fairness (of process schedules or of internal event choices), we believe that it must be performed by coordinators, which mediate component interactions. As a consequence, fairness is a property associated to a coordination purpose and that requires a specific verification, which is out of the scope of this work. Robustness is a desirable property which is not addressed by our work.

Even though there are many approaches to formally model component based systems (CBS) [ADG98, AB03, IM08, HLL06b, PV02], to our knowledge the question of preserving, by construction, behavioural properties has not yet been fully systematised as we have done in this work. Despite the fact that our black-box component contracts are compatible with most component-based approaches, especially those based on CSP or CSP-like notations [Ros98, HLL06b], most approaches to date aim at verifying the entire component-based systems before implementation, but not predicting behavioural properties by construction during design. We can ensure deadlock freedom in a constructive way, as a result of applying composition rules, as opposed to performing model checking verification after the system has been built. The compositional approach can be applied in heterogeneous systems (synchronous and asynchronous) with different topologies (tree or cyclic).

Approaches to verifying a system tend to use abstraction techniques to reduce the state space. They map a set of states of the actual system to an abstract, and a smaller set of states in a way that preserve the behaviours of the system. [ZM10] adopts counterexample guided abstraction refinement scheme to alleviate the state explosion problem of deadlock detection. It extends the classical labelled transition system models by qualifying transitions as certain and uncertain to make deadlock freedom conservative. A similar approach is presented in [Kwi07]. It determines their sets of 'conflict-free' actions, called untangled actions. Untangled actions are compositional; synchronisation on untangled actions will not destroy their 'conflict-freedom'. Following the same approach, [CCH$^+$09] proposes a deadlock detection algorithm based on navigating and marking transitions on a dynamic synchronization dependency graph.

Other approaches tend to design components and interactions using strict component models in order to avoid undesirable properties, such as deadlock. [DZL10] builds up a service interaction model and analyses the deadlock problem related with shared internet resources. It proposes some interaction solutions to effectively prevent deadlocks. In this context, our approach can also be specialised for a specific architectural style. In [RSM09], we combine side conditions presented in this document to propose specific composition rules for interaction components. In this work, all verifications and notions support the analysis of partitions of the component (and composition) behaviour in space (protocols) and time (interaction patterns). This approach combines the advantages of the approaches presented in [VVR06] and in [BBT01], where physical and temporal partitions are realised, respectively. Protocols are observed as a particular type in [VVR06], which permits the verification of compatibility. However, concerns about the entire component behaviour are ignored in the definitions of [VVR06]. Interaction patterns are also defined in [BBT01], however without defining any conformance notion for components or compositions. None of these works defines test characterisations that can mechanically be performed in verification tools.

The study of deadlock freedom is related to the analysis of component incompatibilities. In this context, component compatibility is established by determining those components which, when connected, are free of deadlock. The study of behavioural compatibility helps to reduce the cost of analysing deadlocks in compositions. The criterion exploits compositionality in the sense that a condition is locally checked on pairs of neighbouring components. If the condition is satisfied we can derive the property of deadlock freedom. Thus, the state space construction related complexity is $O(n)$ in the case of the architectural compatibility check, and $O(\alpha^n)$ in the case of the direct check.

In PADL [BCD02] and in [MCM08] compatibility is used to detect architectural mismatches and it is shown that pairwise compatibility is a sufficient criterion to derive deadlock freedom of an acyclic assembly from the deadlock freedom of its local components. These approaches consider the whole behaviour of the constituting components in the composition. Differently, our approach is centred on the use of port protocols to alleviate compatibility verifications.

Closer to our approach is the work presented in [LMC10, CZ07] that performs architectural compatibility verifications based on compatibility of port protocols. The restriction in [LMC10] is that only deterministic protocols are considered. [CZ07] proposes a formal model of component interaction, in which component compatibility is verified using labelled Petri nets. In this work, the behaviour of components is represented solely by their port protocols, called *interface languages*, which contains either possible sequences of required or provided services. A request (rich) interface is compatible with a provider (rich) interface if and only if all sequences of services requested by the former can be provided by the latter. This condition reassembles our denotation definition of compatibility. However, as we deal with bidirectional I/O channels, these conditions are verified in each state of the protocol for both directions.

A notion similar to behavioural compatibility is used by [HJK10b] under the name of *neutrality*. The verification of properties for the whole component then follows from the verification step that uses only weakly deterministic port protocols. Behavioural neutrality is defined in terms of observational equivalence between the behaviour of an assembly with two connected components and the behaviour of an assembly with a single component and the binary connector replaced by a unary one. This notion plays an important role in its reduction strategy. A component neutral to another can be removed from the analysis of composition because they do not contribute with any change in the external observable behaviour of the composition. There are two restrictions in the approach: components must be weakly deterministic and in order to be neutral their input and output labels must mutually coincide. As verified in [CZ07], it is possible that one component does not use all services of another, and, therefore, that one component might output fewer events than the other one may possibly input.

Another notion related to behavioural compatibility is used in [CK96] under the name of *transparency*. In [CK96] automatically derived context constraints (restrictions imposed by the environment on subsystem behaviour) are used to construct the LTS behaviour of composed systems more efficiently.

Context constraints take the form of *interface processes*, which capture the interplay of the environment of a single fixed component as part of the composition with other components. If the composition of the interface process and the fixed process results in a smaller transition system, it is substituted in the overall analysis. The correctness of the approach relies on a transparency property which requires a strong semantic equivalence between the fixed process and its composition with its interface process. Compatibility is verified by checking if the interface process is well-formed. In [All97b], the interface process associated to a port is called a *deterministic process* of a process. Compatibility of two processes is checked by verifying the refinement relationship between a process and the synchronisation of another process and the deterministic process of the former. In our work, the interface process and deterministic versions are called *contextual process*, and similarly to [All97b] is used solely in compatibility checks, rather than in a more general analysis as in [CK96]. Similarly to [All97b], we check compatibility of two protocols as the refinement of a protocol by its context process synchronised with the dual protocol of the other. A dual protocol represents the most nondeterministic process that is compatible with a protocol. We use this notion as we deal with I/O processes in this work.

Our component model considers I/O processes that implicitly support bidirectional communications. The possible existence of non-determinism in I/O processes and of bidirectional communication brings more complexity to our verifications than the works related to the notion of compatibility mentioned above [CK96, All97b, BCD02, MCM08, HJK10b, LMC10]. For instance, in [LMC10, BCD02] components must be deterministic. This prevents designer from considering situations where the components take internal decision (see output decisiveness, Definition 3.12). Bidirectional communication may implicitly introduce small cycles (with two components), and furthermore is not addressed by the works above, since they use compatibility in component-based systems with tree-topology structures of unidirectional channels. However, bidirectional communication is implicit in our component model, and is furthermore directly support by our compatibility notion. Except for the work on PADL [BCD02, AB03], none of the works cited above deal with cyclic topologies. Even this approach does not present a solution to alleviate the verification of applications in such topologies. In [BCD02, AB03] to verify deadlock freedom, deadlock freedom is locally considered in the relationship of each component with the others in the whole cycle. Similarly to the seminal work on deadlock freedom [Ros98], the approach needs to know the internal structure of the entire system (which is also a component) a priori, which is in the opposite direction of a compositional method. In our work, cyclic topologies are verified in compositional correct-by-construction approach, as soon as the cycle appears. A detailed comparison between the basic concepts in [Ros98] and the study on protocol compatibility is presented in [Ram11]. Means to alleviate the verification are presented by the notion of *decoupled channels* (see Section 3).

A further important difference between checking compatibility of port protocols (as done in our work) and checking the compatibility of entire component behaviours is that the use of explicit port behaviours makes the check for compatibility more efficient. Furthermore, as mentioned in [LMC10], this supports a gray box view of the components that is desired in CBD similar to the principle of information hiding. Despite the benefits of port-protocol representation, representing the whole component is also necessary. For instance, the approach in [CZ07] abstracts the internal behaviour of components, and concentrates solely upon the behaviour exhibited by port protocols. Concentrating solely upon the behaviour exhibited by port protocols, these works indirectly restrict the structure of their systems to tree-topologies, without cycles. For the same reason, it is forbidden to assembly multiple points of interaction between components, which implicitly introduce minor cycles. Similarly, the approach forbids the verification of other emerging properties of the system, such as livelock, which emerges from the interaction of the components.

Some approaches [IM08, MW97] do predict some system properties based on the properties of its constituting components. This is performed by categorising components and their communication patterns in order to prevent scenarios in which the interaction among components would introduce improper states. These works focus on different properties. The work reported in [IM08] does not focus on behavioural properties; rather, it presents some results on performance. The approach presented in [MW97] proposes rules to guarantee the absence of deadlocks by construction. However, it presents rules for specific protocol patterns, such as resource sharing and client-server, using simple data communication; for instance, a component must always accept any input data value. As far as we are aware, there is no well-established compositional approach for developing or reasoning about an SoS based on properties of its constituent systems

7.2 Future work (Deliverable 24.4)

The correct-by-construction approach proposed in this document can be extended in many ways. There are several directions for building on the results of this work. Moreover, there are opportunities for new interesting related research directions. Here we focus on the extensions that are planned for Deliverable D24.4 (due in Month 36). They are as follows.

More complex Case Studies. In order to better support the process of SoS development, it is necessary to develop further case studies and carefully analyse what is the support needed by the developers to apply the proposed rules. The case studies will also contribute to identify architectural patterns for systems of systems.

Evaluation of the time complexity of the approach. An important issue is to perform a comparative study of the performance of our approach and other existing approaches in the literature as that presented in Section 3.4, but in the context of CML and its model-checker, which is currently under development. This is essential to reinforce the benefits of our approach. More specifically, dealing with time complexity issues is essential for modelling and analysing large systems of systems.

Increased breadth of architectural styles support. This is both in the number of styles and in the specialised constructive constraints to support their development. This requires the study of the specialities of each style and how these would help to alleviate verifications in a compositional approach, particularly considering the reflexive composition rule.

Adopt a concrete syntax. It is essential to adopt a more convenient concrete syntax to the use of our notions. SysML [FMS08] introduces notions of components, ports and structured classifiers which are, not surprisingly, a perfect match with the syntactic requirements of our component model. Within the COMPASS project, SysML has been given a CML semantics. This allows us to try and adopt the approach at the SysML level.

Incorporate new metadata to enrich component contracts that can improve our approach. Other metadata can be identified and incorporated to our approach. We strongly believe that this will be needed for further optimisations to the approach. For instance, by getting rid of some additional side conditions we might improve even more the verification time of our approach. One of the conditions related to our rule on *reflexive composition*, which allows the connection of two independent channels of a same component, is the only non-compositional condition. The use of properties that are inherent to architectural styles may be used to address some of these conditions. For instance, a star topology would not require the use of the reflexive composition. A thorough analysis of the remaining side conditions and architectural styles is in our research agenda for Deliverable 24.4.

Incorporating livelock treatment (Wrapping operators). In this document, we focused on assuring only deadlock freedom by construction. Nevertheless, in [Ram11], we have presented a means to guarantee also livelock-freedom by construction by providing *wrappings* to perform *safe hidings*. The lift of this result to CML is an interesting point of further investigation.

Substitutability. Besides composition, substitution is another important aspect in the development of SoSs. Most works on substitutability are based on the notions of behavioural subtyping [LW94, Weh03], which is a strong form of relationship between two (component) types. It requires instances of a *subtype* and of a *supertype* to fulfil the principle of *type substitutability* [LW94]:

> An instance of the subtype should be usable wherever an instance of the supertype is expected, without a client being able to tell the difference.

This suggests the use of some form of *refinement* [Ros98] to formalise behavioural subtyping. Refinement guarantees substitutability in an even stronger form: a system can always be replaced by its refinement without any noticeable difference. For subtyping, we want only a replacement to be unnoticeable at places where *a supertype is expected*. This is a weaker form of substitutability, but that nevertheless can be characterised in terms of refinement [Weh03]. Different substitutability relations can be defined if we are aware of the context in which the component is.

In contrast with composition, substitutability relates constituents that are not currently presented in the system (it relates a present configuration with a future one). As a consequence, besides the definition of substitutability notions, it is also necessary to establish its relation with other constructive relations, as the composition rules presented here.

Service conformance Service conformance can be understood as a design principle to be followed: unused services of a component should be still available after composition. The degree of satisfaction of this notion may vary from preserving all services (strong conformance) to at least one (weak conformance). In principle, it is easy to characterise these notions in our CSP setting, by projecting the behaviour of the composed system and comparing with the behaviour of each constituent. Nevertheless, this projection involves hiding and, as already discussed, can lead to divergent behaviour. So, more investigation is necessary, particularly in the context of SoS.

Appendix

A Refinement Laws

The following refinement laws are taken from [Oli06] and [OZC11].

Law 1 (var-exp-par)

$$(\mathbf{var}\ d : T \bullet A_1) \llbracket ns_1 \mid cs \mid ns_2 \rrbracket A_2 = (\mathbf{var}\ d : T \bullet A_1 \llbracket ns_1 \mid cs \mid ns_2 \rrbracket A_2)$$

provided $\{ d, d' \} \cap FV(A_2) = \emptyset$

Law 2 (var-exp-par-2)

$$(\mathbf{var}\ d \bullet A_1) \llbracket ns_1 \mid cs \mid ns_2 \rrbracket (\mathbf{var}\ d \bullet A_2) = (\mathbf{var}\ d \bullet A_1 \llbracket ns_1 \mid cs \mid ns_2 \rrbracket A_2)$$

Law 3 (var-exp-rec) $\mu X \bullet (\mathbf{var}\ x : T \bullet F(X)) = \mathbf{var}\ x : T \bullet (\mu X \bullet F(X))$

provided *x is initialised before use in F*

Law 4 (var-exp-seq) $A_1;(\mathbf{var}\ x : T \bullet A_2);A_3 = (\mathbf{var}\ x : T \bullet A_1;A_2;A_3)$

provided $\{ x, x' \} \cap (FV(A_1) \cup FV(A_3)) = \emptyset$

Law 5 (Variable Substitution)

$$A(x) = \mathbf{var}\ y \bullet y : [y' = x];\ A(y)$$

provided *y is not free in A*

Law 6 (Variable block introduction*)

$$A = \mathbf{var}\ x : T \bullet A$$

provided $x \notin FV(A)$

Law 7 (join-blocks) $\mathbf{var}\ x : T_1 \bullet \mathbf{var}\ y : T_2 \bullet A = \mathbf{var}\ x : T_1;\ y : T_2 \bullet A$

Law 8 (Sequence unit)

$$(A) Skip;\ A = A$$
$$(B) A = A;\ Skip$$

Law 9 (Recursion unfold)

$$\mu X \bullet F(X) = F(\mu X \bullet F(X))$$

Law 10 (Parallelism composition/External choice—expansion)

$$\begin{array}{l}(\Box\, i \bullet a_i \to A_i) \llbracket ns_1 \mid cs \mid ns_2 \rrbracket (\Box\, j \bullet b_j \to B_j) \\ = \\ (\Box\, i \bullet a_i \to A_i) \llbracket ns_1 \mid cs \mid ns_2 \rrbracket ((\Box\, j \bullet b_j \to B_j) \Box (c \to C))\end{array}$$

provided

- $\bigcup_i \{a_i\} \subseteq cs$
- $c \in cs$
- $c \notin \bigcup_i \{a_i\}$
- $c \notin \bigcup_j \{b_j\}$

Law 11 (Parallelism composition introduction 1)

$$c \to A = (c \to A \llbracket ns_1 \mid \{\!|\ c\ |\!\} \mid ns_2 \rrbracket c \to Skip)$$
$$c.e \to A = (c.e \to A \llbracket ns_1 \mid \{\!|\ c\ |\!\} \mid ns_2 \rrbracket c.e \to Skip)$$

provided

- $c \notin usedC(A)$
- $wrtV(A) \subseteq ns_1$

Law 12 (Channel extension 1)

$$A_1 \llbracket ns_1 \mid cs \mid ns_2 \rrbracket A_2 = A_1 \llbracket ns_1 \mid cs \cup \{\!|c|\!\} \mid ns_2 \rrbracket A_2$$

provided $c \notin usedC(A_1) \cup usedC(A_2)$

Law 13 (Hiding expansion 2)

$$A \setminus cs = A \setminus cs \cup \{c\}$$

provided $c \notin usedC(A)$

Law 14 (Prefix/Hiding)

$$(c \to Skip) \setminus \{c\} = Skip$$
$$(c.e \to Skip) \setminus \{c\} = Skip$$

Law 15 (Hiding Identity)

$$A \setminus cs = A$$

provided $cs \cap usedC(A) = \emptyset$

Law 16 (Parallelism composition/External choice—exchange)

$$(A_1 \llbracket ns_1 \mid cs \mid ns_2 \rrbracket A_2) \Box (B_1 \llbracket ns_1 \mid cs \mid ns_2 \rrbracket B_2)$$
$$=$$
$$(A_1 \Box B_1) \llbracket ns_1 \mid cs \mid ns_2 \rrbracket (A_2 \Box B_2)$$

provided $A_1 \llbracket ns_1 \mid cs \mid ns_2 \rrbracket B_2 = A_2 \llbracket ns_2 \mid cs \mid ns_1 \rrbracket B_1 = Stop$

Law 17 (Parallelism composition/External choice—distribution*)

$$\Box\, i \bullet (A_i \llbracket ns_1 \mid cs \mid ns_2 \rrbracket A) = (\Box\, i \bullet A_i) \llbracket ns_1 \mid cs \mid ns_2 \rrbracket A$$

provided
- $initials(A) \subseteq cs$
- A *is deterministic*

Law 18 (External choice unit)

$$Stop \Box A = A$$

Law 19 (External choice/Sequence—distribution)

$$(\Box\, i \bullet g_i \,\&\, c_i \to A_i);\; B = \Box\, i \bullet g_i \,\&\, c_i \to A_i;\; B$$

Law 20 (Hiding/External choice—distribution)

$$(A_1 \Box A_2) \setminus cs = (A_1 \setminus cs) \Box (A_2 \setminus cs)$$

provided $(\mathit{initials}(A_1) \cup \mathit{initials}(A_2)) \cap cs = \emptyset$

Law 21 (Parallelism composition Deadlocked 1)

$$(c_1 \to A_1) \llbracket ns_1 \mid cs \mid ns_2 \rrbracket (c_2 \to A_2) = \mathit{Stop} = \mathit{Stop} \llbracket ns_1 \mid cs \mid ns_2 \rrbracket (c_2 \to A_2)$$

provided
- $c_1 \neq c_2$
- $\{c_1, c_2\} \subseteq cs$

Law 22 (Sequence zero)

$$\mathit{Stop};\ A = \mathit{Stop}$$

Law 23 (Communication/Parallelism composition—distribution)

$$(c!e \to A_1) \llbracket ns_1 \mid cs \mid ns_2 \rrbracket (c?x \to A_2(x)) = c.e \to (A_1 \llbracket ns_1 \mid cs \mid ns_2 \rrbracket A_2(e))$$

provided
- $c \in cs$
- $x \notin FV(A_2)$.

Law 24 (Channel extension 3*)

$$\begin{array}{l}(A_1 \llbracket ns_1 \mid cs_1 \mid ns_2 \rrbracket A_2(e)) \setminus cs_2 \\ = \\ ((c!e \to A_1) \llbracket ns_1 \mid cs_1 \mid ns_2 \rrbracket (c?x \to A_2(x))) \setminus cs_2\end{array}$$

provided
- $c \in cs_1$
- $c \in cs_2$
- $x \notin FV(A_2)$

Law 25 (Channel extension 4*)

$$(A_1 \llbracket ns_1 \mid cs_1 \mid ns_2 \rrbracket A_2) \setminus cs_2 = ((c \rightarrow A_1) \llbracket ns_1 \mid cs_1 \mid ns_2 \rrbracket (c \rightarrow A_2)) \setminus cs_2$$

$$(A_1 \llbracket ns_1 \mid cs_1 \mid ns_2 \rrbracket A_2) \setminus cs_2 = ((c.e \rightarrow A_1) \llbracket ns_1 \mid cs_1 \mid ns_2 \rrbracket (c.e \rightarrow A_2)) \setminus cs_2$$

provided

- $c \in cs_1$
- $c \in cs_2$

The reference to $\mathbf{L}(\_)$ denotes the fact that declarations of x (and x') in schemas, which were used to put the local variable x of the main action into scope, may now be removed, as x is a state component.

Law 26 (prom-var-state)

$$\textbf{begin}\ (\textbf{state}\ S)\ \mathbf{L}(x : T) \bullet (\textbf{var}\ x : T \bullet \mathbf{MA})\ \textbf{end}$$
$$=$$
$$\textbf{begin}\ (\textbf{state}\ S \land [\,x : T\,])\ \mathbf{L}(\_) \bullet \mathbf{MA}\ \textbf{end}$$

Law 27 (prom-var-state-2)

$$\textbf{begin}\ \mathbf{L}(x : T) \bullet (\textbf{var}\ x : T \bullet \mathbf{MA})\ \textbf{end}$$
$$=$$
$$\textbf{begin}\ (\textbf{state}\ [\,x : T\,])\ \mathbf{L}(\_) \bullet \mathbf{MA}\ \textbf{end}$$

Law 28 (Parallelism composition unit)

$$Skip \llbracket ns_1 \mid cs \mid ns_2 \rrbracket Skip = Skip$$

Law 29 (Parallelism composition/Interleaving Equivalence)

$$A_1 \parallel\!\![ns_2 \mid ns_2]\!\!\parallel A_2 = A_1 \llbracket ns_2 \mid \emptyset \mid ns_2 \rrbracket A_2$$

Law 30 (Parallelism composition/Sequence—step)

$$(A_1;\ A_2) \llbracket\, ns_1 \mid cs \mid ns_2 \,\rrbracket A_3 = A_1;\ (A_2 \llbracket\, ns_1 \mid cs \mid ns_2 \,\rrbracket A_3)$$

provided

- $\mathit{initials}(A_3) \subseteq cs$
- $cs \cap \mathit{usedC}(A_1) = \emptyset$
- $\mathit{wrtV}(A_1) \cap \mathit{usedV}(A_3) = \emptyset$
- A_3 *is divergence-free*
- $\mathit{wrtV}(A_1) \subseteq ns_1$

Law 31 (Hiding/Sequence—distribution*)

$$(A_1;\ A_2) \setminus cs = (A_1 \setminus cs);\ (A_2 \setminus cs)$$

Law 32 (Guard/Sequence—associativity)

$$(g \ \&\ A_1);\ A_2 = g \ \&\ (A_1;\ A_2)$$

Law 33 (Input prefix/Parallelism composition—distribution 2*)

$$c?x \to (A_1 \llbracket\, ns_1 \mid cs \mid ns_2 \,\rrbracket A_2) = (c?x \to A_1) \llbracket\, ns_1 \mid cs \mid ns_2 \,\rrbracket A_2$$

provided

- $c \notin cs$
- $x \notin \mathit{usedV}(A_2)$
- $\mathit{initials}(A_2) \subseteq cs$
- A_2 *is deterministic*

Law 34 (Prefix/*Skip)**

$$c \to A = (c \to \mathit{Skip});\ A$$

$$c.e \to A = (c.e \to \mathit{Skip});\ A$$

Law 35 (Prefix/Parallelism composition—distribution)

$$c \to (A_1 \llbracket\, ns_1 \mid cs \mid ns_2 \,\rrbracket A_2) = (c \to A_1) \llbracket\, ns_1 \mid cs \cup \{\!|c|\!\} \mid ns_2 \,\rrbracket (c \to A_2)$$

$$c.e \to (A_1 \llbracket\, ns_1 \mid cs \mid ns_2 \,\rrbracket A_2) = (c.e \to A_1) \llbracket\, ns_1 \mid cs \cup \{\!|c|\!\} \mid ns_2 \,\rrbracket (c.e \to A_2)$$

provided $c \notin \mathit{usedC}(A_1) \cup \mathit{usedC}(A_2)$ *or* $c \in cs$

Law 36 (External choice/Sequence—distribution 2*)

$$((g_1 \mathbin{\&} A_1) \mathbin{\Box} (g_2 \mathbin{\&} A_2));\ B = ((g_1 \mathbin{\&} A_1);\ B) \mathbin{\Box} ((g_2 \mathbin{\&} A_2);\ B)$$

provided $g_1 \Rightarrow \neg\, g_2$

Law 37 (True guard)

$$\mathit{true} \mathbin{\&} A = A$$

Law 38 (False guard)

$$\mathit{false} \mathbin{\&} A = \mathit{Stop}$$

Law 39 (Hiding/Chaos–distribution)

$$\mathit{Chaos} \setminus cs = \mathit{Chaos}$$

Law 40 (Sequence zero 2)

$$\mathit{Chaos};\ A = \mathit{Chaos}$$

Law 41 (Parallelism composition Zero)

$$\mathit{Chaos} \mathbin{[\![} ns_1 \mid cs \mid ns_2 \mathbin{]\!]} A = \mathit{Chaos}$$

Law 42 (Internal choice/Parallelism composition Distribution)

$$\begin{array}{l}(A_1 \sqcap A_2) \mathbin{[\![} ns_1 \mid cs \mid ns_2 \mathbin{]\!]} A_3 \\ = \\ (A_1 \mathbin{[\![} ns_1 \mid cs \mid ns_2 \mathbin{]\!]} A_3) \sqcap (A_2 \mathbin{[\![} ns_1 \mid cs \mid ns_2 \mathbin{]\!]} A_3)\end{array}$$

Law 43 (Sequence/Internal choice—distribution)

$$A_1;\ (A_2 \sqcap A_3) = (A_1;\ A_2) \sqcap (A_1;\ A_3)$$

Law 44 (Hiding/Parallelism composition—distribution*)

$$(A_1 \llbracket ns_1 \mid cs_1 \mid ns_2 \rrbracket A_2) \setminus cs_2 = (A_1 \setminus cs_2) \llbracket ns_1 \mid cs_1 \mid ns_2 \rrbracket (A_2 \setminus cs_2)$$

provided $cs_1 \cap cs_2 = \emptyset$

Law 45 (Hiding combination)

$$(A \setminus cs_1) \setminus cs_2 = A \setminus (cs_1 \cup cs_2)$$

The following refinement laws are novel and a further contribution of this document.

Law 46 (Parallelism composition Deadlocked 3*)

$$\begin{aligned}&(\Box_i\, c_i \to A_i) \llbracket ns_1 \mid cs \mid ns_2 \rrbracket (\Box_j\, c_j \to A_j)\\ &= Stop\\ &= Stop \llbracket ns_1 \mid cs \mid ns_2 \rrbracket (\Box_j\, c_j \to A_j)\end{aligned}$$

provided
- $\bigcup_i \{c_i\} \cap \bigcup_j \{c_j\} = \emptyset$
- $\bigcup_i \{c_i\} \cup \bigcup_j \{c_j\} \subseteq cs$

Law 47 (Assignment Removal*)

$$x := e;\ A(x) = A(e)$$

provided x *is not free in* $A(e)$

Law 48 (Innocuous Assignment*)

$$x := x = Skip$$

Law 49 (Variable Substitution 2*)

$$\mathbf{var}\ x \bullet A(x) = \mathbf{var}\ y \bullet A(y)$$

provided
- $x \notin FV(A(y))$
- $y \notin FV(A(x))$

Law 50 (Input Prefix/Sequence Distribution*)

$$(c?x \rightarrow A_1);\, A_2 = c?x \rightarrow (A_1;\, A_2)$$

provided $x \notin FV(A_2)$

Law 51 (Input Prefix/Hiding Identity*)

$$(c?x \rightarrow A) \setminus cs = c?x \rightarrow (A \setminus cs)$$

provided $c \notin cs$

Law 52 (Guard/Parallelism composition–distribution*))

$$(g \,\&\, A1) \llbracket ns_1 \mid cs \mid ns_2 \rrbracket A2 = g \,\&\, (A1 \llbracket ns_1 \mid cs \mid ns_2 \rrbracket A2)$$

provided

- $(initials(A2) \subseteq cs)$

Law 53 (Internal choice/Hiding composition Distribution)

$$(A1 \sqcap A2) \setminus cs = (A1 \setminus cs) \sqcap (A2 \setminus cs)$$

Law 54 (Alternation Zero)

$$\begin{array}{l}\textbf{if} \parallel i \bullet g_i \rightarrow\ A_i \textbf{ fi} \\ = \\ Chaos\end{array}$$

provided $\bigvee i \bullet g_i \equiv false$

Law 55 (Alternation)

$$\begin{array}{l}\textbf{if} \parallel i : S \bullet g_i \rightarrow\ A_i \textbf{ fi} \\ = \\ \bigsqcap i : T \bullet A_i\end{array}$$

provided

- $T \subseteq S$
- $\bigwedge i : T \bullet g_i \equiv true$
- $\bigvee i : S \setminus T \bullet g_i \equiv false$

Law 56 (Assignment Skip)

$$
\begin{array}{l}
\mathbf{var}\ x \bullet x := e \\
= \\
\mathbf{var}\ x \bullet \mathit{Skip}
\end{array}
$$

B Mapping Functions

B.1 Mapping Function for Actions

The mapping function is defined as follows:

$$
\begin{array}{l}
\Upsilon\,(\mathit{Skip}) \mathrel{\widehat=} \texttt{SKIP} \\
\Upsilon\,(\mathit{Stop}) \mathrel{\widehat=} \texttt{STOP} \\
\Upsilon\,(c \to \mathit{Skip}) \mathrel{\widehat=} \texttt{c} \to \texttt{SKIP} \\
\Upsilon\,(c \to A) \mathrel{\widehat=} \texttt{c} \to \Upsilon(A) \\
\Upsilon\,(c.v \to A) \mathrel{\widehat=} \texttt{c.v} \to \Upsilon(A) \\
\Upsilon\,(c!v \to A) \mathrel{\widehat=} \texttt{c!v} \to \Upsilon(A) \\
\Upsilon\,(c?x : P \to A) \mathrel{\widehat=} \texttt{c?x} : \{\texttt{x} \mid \texttt{x} \leftarrow \delta(c), \Upsilon_{\mathbb{B}}(P(x))\} \to \Upsilon(A) \\
\Upsilon\,(c?x \to A) \mathrel{\widehat=} \texttt{c?x} \to \Upsilon(A) \\
\Upsilon\,(A \mathbin{\Box} B) \mathrel{\widehat=} \Upsilon(A) \mathbin{\Box} \Upsilon(B) \\
\Upsilon\,(A \sqcap B) \mathrel{\widehat=} \Upsilon(A) \sqcap \Upsilon(B) \\
\Upsilon\,(g \mathbin{\&} A) \mathrel{\widehat=} \Upsilon_{\mathbb{B}}(g) \mathbin{\&} \Upsilon(A) \\
\Upsilon\,(A; B) \mathrel{\widehat=} \Upsilon(A); \Upsilon(B) \\
\Upsilon\,(A \mathbin{[\![} ns_1 \mid cs \mid ns_2 \mathbin{]\!]} B) \mathrel{\widehat=} \Upsilon(A)[|\Upsilon_{\mathbb{P}^{cs}}(cs)|]\Upsilon(B) \\
\Upsilon\,(A \mathbin{|\!|\![} ns_1 \mid ns_2 \mathbin{]\!|\!|} B) \mathrel{\widehat=} \Upsilon(A)|||\Upsilon(B) \\
\Upsilon\,(A \setminus cs) \mathrel{\widehat=} \Upsilon(A)\setminus\Upsilon_{\mathbb{P}^{cs}}(cs) \\
\Upsilon\,(\Box\, x : S \bullet A) \mathrel{\widehat=} \Box\, x : \Upsilon_{\mathbb{P}}(S) \bullet \Upsilon(A) \\
\Upsilon\,(\bigsqcap x : S \bullet A) \mathrel{\widehat=} \bigsqcap x : \Upsilon_{\mathbb{P}}(S) \bullet \Upsilon(A) \\
\Upsilon\,(\mathbin{;}\ x : S \bullet A) \mathrel{\widehat=} ;\ x : \Upsilon_{\mathsf{seq}}(S) \bullet \Upsilon(A) \\
\Upsilon\,([\![cs]\!]\ x : S \bullet [\![\emptyset]\!]\ A) \mathrel{\widehat=} [|\Upsilon_{\mathbb{P}^{cs}}(cs)|]\ \texttt{x} : \Upsilon_{\mathbb{P}}(S) \bullet \Upsilon(A) \\
\Upsilon\,(|||\ x : S \bullet |\!|\![\emptyset]\!|\!|\ A) \mathrel{\widehat=} |||\ \texttt{x} : \Upsilon_{\mathbb{P}}(S) \bullet \Upsilon(A) \\
\Upsilon\,(\mu\, X \bullet A(X)) \mathrel{\widehat=} \texttt{let}\ \texttt{A}_{\texttt{rec}} = \Upsilon(A(A_{rec}))\ \texttt{within}\ \texttt{A}_{\texttt{rec}}
\end{array}
$$

B.2 Mapping Function for Numbers

The mapping function for set expressions is defined as follows:

$\Upsilon_{\mathbb{Z}}(n) \mathrel{\widehat=} \texttt{n}$
$\Upsilon_{\mathbb{Z}}(0) \mathrel{\widehat=} \texttt{0}\ldots$
$\Upsilon_{\mathbb{Z}}(n+m) \mathrel{\widehat=} \Upsilon_{\mathbb{Z}}(n)\texttt{+}\Upsilon_{\mathbb{Z}}(m)$
$\Upsilon_{\mathbb{Z}}(n-m) \mathrel{\widehat=} \Upsilon_{\mathbb{Z}}(n)\texttt{-}\Upsilon_{\mathbb{Z}}(m)$
$\Upsilon_{\mathbb{Z}}(-n) \mathrel{\widehat=} -\Upsilon_{\mathbb{Z}}(m)$
$\Upsilon_{\mathbb{Z}}(n*m) \mathrel{\widehat=} \Upsilon_{\mathbb{Z}}(n)*\Upsilon_{\mathbb{Z}}(m)$
$\Upsilon_{\mathbb{Z}}(n \,\mathsf{div}\, m) \mathrel{\widehat=} \Upsilon_{\mathbb{Z}}(n)/\Upsilon_{\mathbb{Z}}(m)$
$\Upsilon_{\mathbb{Z}}(n \,\mathsf{mod}\, m) \mathrel{\widehat=} \Upsilon_{\mathbb{Z}}(n)\%\Upsilon_{\mathbb{Z}}(m)$

B.3 Mapping Function for Predicates

The mapping function for predicates is defined as follows:

$\Upsilon_{\mathbb{B}}(n) \mathrel{\widehat=} \texttt{n}$
$\Upsilon_{\mathbb{B}}(\mathit{true}) \mathrel{\widehat=} \texttt{true}$
$\Upsilon_{\mathbb{B}}(\mathit{false}) \mathrel{\widehat=} \texttt{false}$
$\Upsilon_{\mathbb{B}}(a \wedge b) \mathrel{\widehat=} \Upsilon_{\mathbb{B}}(a)\,\texttt{and}\,\Upsilon_{\mathbb{B}}(b)$
$\Upsilon_{\mathbb{B}}(a \vee b) \mathrel{\widehat=} \Upsilon_{\mathbb{B}}(a)\,\texttt{or}\,\Upsilon_{\mathbb{B}}(b)$
$\Upsilon_{\mathbb{B}}(\neg\, a) \mathrel{\widehat=} \texttt{not}\,\Upsilon_{\mathbb{B}}(a)$
$\Upsilon_{\mathbb{B}}(a = b) \mathrel{\widehat=} \Upsilon(a) = \Upsilon(b)$
$\Upsilon_{\mathbb{B}}(a \neq b) \mathrel{\widehat=} \Upsilon(a)\,! = \Upsilon(b)$
$\Upsilon_{\mathbb{B}}(a < b) \mathrel{\widehat=} \Upsilon_{\mathbb{Z}}(a) < \Upsilon_{\mathbb{Z}}(b)$
$\Upsilon_{\mathbb{B}}(a \leq b) \mathrel{\widehat=} \Upsilon_{\mathbb{Z}}(a) <= \Upsilon_{\mathbb{Z}}(b)$
$\Upsilon_{\mathbb{B}}(a > b) \mathrel{\widehat=} \Upsilon_{\mathbb{Z}}(a) > \Upsilon_{\mathbb{Z}}(b)$
$\Upsilon_{\mathbb{B}}(a \geq b) \mathrel{\widehat=} \Upsilon_{\mathbb{Z}}(a) >= \Upsilon_{\mathbb{Z}}(b)$
$\Upsilon_{\mathbb{B}}(\textbf{if}\ \ b\ \textbf{then}\ \ x1\ \textbf{else}\ \ x2) \mathrel{\widehat=} \texttt{if}\,\Upsilon(b)\,\texttt{then}\,\Upsilon(x1)\,\texttt{else}\,\Upsilon(x2)$

B.4 Mapping Function for Set Expressions

The mapping function for set expressions is defined as follows:

$\Upsilon_{\mathbb{P}}\,(n) \mathrel{\widehat{=}} \mathtt{n}$
$\Upsilon_{\mathbb{P}}\,(\{\}) \mathrel{\widehat{=}} \{\}$
$\Upsilon_{\mathbb{P}}\,(\{a, \ldots, b\}) \mathrel{\widehat{=}} \{\Upsilon(a), \ldots, \Upsilon(a)\}$
$\Upsilon_{\mathbb{P}}\,(n \mathinner{\ldotp\ldotp} m) \mathrel{\widehat{=}} \{\Upsilon(n)..\Upsilon(m)\}$
$\Upsilon_{\mathbb{P}}\,(a \cup b) \mathrel{\widehat{=}} \mathtt{union}(\Upsilon_{\mathbb{P}}(a),\Upsilon_{\mathbb{P}}(b))$
$\Upsilon_{\mathbb{P}}\,(a \cap b) \mathrel{\widehat{=}} \mathtt{inter}(\Upsilon_{\mathbb{P}}(a),\Upsilon_{\mathbb{P}}(b))$
$\Upsilon_{\mathbb{P}}\,(a \setminus b) \mathrel{\widehat{=}} \mathtt{diff}(\Upsilon_{\mathbb{P}}(a),\Upsilon_{\mathbb{P}}(b))$
$\Upsilon_{\mathbb{P}}\,(\bigcup A) \mathrel{\widehat{=}} \mathtt{Union}(\Upsilon_{\mathbb{P}}(A))$
$\Upsilon_{\mathbb{P}}\,(\bigcap A) \mathrel{\widehat{=}} \mathtt{Inter}(\Upsilon_{\mathbb{P}}(A))$
$\Upsilon_{\mathbb{P}}\,(x \in A) \mathrel{\widehat{=}} \mathtt{member}(\Upsilon_{\mathbb{P}}(x),\Upsilon_{\mathbb{P}}(A))$
$\Upsilon_{\mathbb{P}}\,(\#A) \mathrel{\widehat{=}} \mathtt{card}(\Upsilon_{\mathbb{P}}(a))$
$\Upsilon_{\mathbb{P}}\,(\operatorname{ran} s) \mathrel{\widehat{=}} \mathtt{set}(\Upsilon_{\mathbb{P}}(s))$
$\Upsilon_{\mathbb{P}}\,(\mathbb{P}\,A) \mathrel{\widehat{=}} \mathtt{Set}(\Upsilon_{\mathbb{P}}(A))$
$\Upsilon_{\mathbb{P}}\,(\operatorname{seq} A) \mathrel{\widehat{=}} \mathtt{Seq}(\Upsilon_{\mathbb{P}}(A))$
$\Upsilon_{\mathbb{P}}\,(\{x_1 : a_1;\ \ldots;\ x_n : a_n \mid b \bullet E(x_1, ..., x_n)\}) \mathrel{\widehat{=}}$
$\quad \{\Upsilon(E(x_1, ..., x_n))|\Upsilon(xi) \leftarrow \Upsilon(ai),\Upsilon(b)\}$

B.5 Mapping Function for Channel Set Expressions

The mapping function for channel set expressions is defined as follows:

$\Upsilon_{\mathbb{P}^{cs}}\,(cs) \mathrel{\widehat{=}} \bigcup\{\{\!|\ c\ |\!\} \mid c \leftarrow \Upsilon_{\mathbb{P}}(cs)\}$

B.6 Mapping Function for Sequence Expressions

The mapping function for sequence expressions is defined as follows:

$\Upsilon_{\mathrm{seq}}\,(n) \mathrel{\widehat{=}} \mathtt{n}$
$\Upsilon_{\mathrm{seq}}\,(\langle\rangle) \mathrel{\widehat{=}} \langle\rangle$
$\Upsilon_{\mathrm{seq}}\,(\langle a, \ldots, b\rangle) \mathrel{\widehat{=}} \langle\Upsilon(a), \ldots, \Upsilon(b)\rangle$
$\Upsilon_{\mathrm{seq}}\,(s \frown t) \mathrel{\widehat{=}} \Upsilon_{\mathrm{seq}}(s)^{\Upsilon}_{\mathrm{seq}}(t)$
$\Upsilon_{\mathrm{seq}}\,(\#s) \mathrel{\widehat{=}} \#(\Upsilon_{\mathrm{seq}}(s))$
$\Upsilon_{\mathrm{seq}}\,(\mathit{head}(s)) \mathrel{\widehat{=}} \mathtt{head}(\Upsilon_{\mathrm{seq}}(s))$
$\Upsilon_{\mathrm{seq}}\,(\mathit{tail}(s)) \mathrel{\widehat{=}} \mathtt{tail}(\Upsilon_{\mathrm{seq}}(s))$
$\Upsilon_{\mathrm{seq}}\,(\frown/(S)) \mathrel{\widehat{=}} \mathtt{concat}(\Upsilon_{\mathrm{seq}}(S))$

C Prefixed Actions

Definition C.1 (Prefixed Actions) *Prefixed actions are initially allowed only to synchronise on some event. They have one the following structure:*

- A, *where the definition of* A *is a* ***Circus*** *prefixed action;*
- $A[old_0, \ldots, old_n := new_0, \ldots, new_n]$, *where the definition of* A *is a* ***Circus*** *prefixed action;*
- $c \rightarrow A$, *where* c *has any communication structure allowed by* ***Circus****;*
- $g \mathbin{\&} A$, *where* A *is a* ***Circus*** *prefixed action;*
- $A_1 \mathbin{;} A_2$, *where* A_1 *is a* ***Circus*** *prefixed action;*
- $A_1 \mathbin{\Box} A_2$, *where* A_1 *and* A_2 *are* ***Circus*** *prefixed actions;*
- $A_1 \sqcap A_2$, *where* A_1 *and* A_2 *are* ***Circus*** *prefixed actions;*
- $A_1 \llbracket ns_1 \mid cs \mid ns_2 \rrbracket A_2$, *where* A_1 *and* A_2 *are* ***Circus*** *prefixed actions;*
- $A_1 \mathbin{|||[} ns_1 \mid cs \mid ns_2 \mathbin{]|||} A_2$, *where* A_1 *and* A_2 *are* ***Circus*** *prefixed actions;*
- $A \setminus cs$, *where* A *is a* ***Circus*** *prefixed action and* $initials(A) \cap cs = \emptyset$*;*
- $(x : T \bullet A(x))(e)$, *where* A *is a* ***Circus*** *prefixed action;*
- $\mu X \bullet A(X)$, *where* A *is a* ***Circus*** *prefixed action;*
- $\fatsemi\ x : \langle v_1, \ldots, v_n \rangle \bullet A(x)$, *where* $A(v_1)$ *is a* ***Circus*** *prefixed action*
- $\Box\, x : T \bullet A(x)$, *where* $A(x)$ *is a* ***Circus*** *prefixed action, for all* $x : T$
- $\sqcap\, x : T \bullet A(x)$, *where* $A(x)$ *is a* ***Circus*** *prefixed action, for all* $x : T$
- $\llbracket\, cs \,\rrbracket\, x : \{v_1, \ldots, v_n\} \bullet \rrbracket\, ns(x)\, \rrbracket\, A(x)$, *where* $A(x)$ *is a* ***Circus*** *prefixed action, for all* $x : T$
- $|||\, cs \,\rrbracket\, x : \{v_1, \ldots, v_n\} \bullet ||[\, ns(x)\,]||\, A(x)$, *where* $A(x)$ *is a* ***Circus*** *prefixed action, for all* $x : T$
- **if** $g_0 \rightarrow A_0 \,[\!]\, \ldots \,[\!]\, g_n \rightarrow A_n$ **fi**, *where* A_i *is a* ***Circus*** *prefixed action, for all* $i : 0 \ldots n$
- **var** $decl \bullet A$, *where* A *is a* ***Circus*** *prefixed action*

D *RingBuffer*: from CML to CSP

D.1 CML *RingBuffer*

```
types
    Value = nat
    CellId = nat inv id == id > 0 and id <= maxring
    Direction =  <req> | <ack>

values
    maxbuff = 4;
    maxring = maxbuff - 1
    Ctr_I = { rd_i, wrt_i }

channels
    input, output : Value
    write, read: CellId * Direction * Value
    rrd, wrt: Direction * Value
    rd_i, wrt_i: CellId * Direction * Value


process RingCell =
begin
    state v:Value
    operations
        setV(x:Value)
            frame wr v
            post v = x
    actions
        Act = wrt.req?x -> setV(x); wrt.ack.x -> Act
              []
              rrd.req?dumb -> rrd.ack!v -> Act
    @ Act
end

process IRCell =
    i:CellId @ RingCell [[ rrd <- rd_i.i, wrt <- wrt_i.i]]

process DRing = ||| i: CellId @ IRCell(i)
process Controller =
begin
    state cache:Value;
          size:nat;
          top:CellId;
          bot:CellId
    operations
        Init(c:Value, s:nat, t:CellId, b:CellId)
            post cache=c and size=s and top=t and bot=b

        SetCache(x:Value)
            frame wr cache:Value
            post cache = x

        SetSize(x:nat)
            frame wr size:nat
            post size = x

        SetTop(x:CellId)
            frame wr top:CellId
            post top = x

        SetBot(x:CellId)
            frame wr bot:CellID
            post bot = x

    actions
        Input =
            [size < maxbuff] &
                input?x ->
                    ( [size = 0] & SetCache(x); SetSize(1)
                      []
                      [size > 0] &
                            write.top.req!x ->
                            write.top.ack?dumb ->
                            SetSize(size+1);
                            SetTop((top mod maxring)+1) )
        Output =
            [size > 0] &
                output!cache ->
                    ( [size > 1] &
                        (|~| dumb:Value @
                            read.bot.req.dumb ->
                            read.bot.ack?x -> SetCache(x));
                        SetSize(size-1);
                        SetBot((bot mod maxring)+1)
                      []
                      [size = 1] &
                        SetSize(0))

    @ Init(0,0,1,1); mu X @ ((Input [] Output); X)
end

process ControllerR =
    Controller [[ read <- rd_i, write <- wrt_i ]]

process DBuffer = (ControllerR [| Ctr_I |] DRing) \ Ctr_I
```

D.2 *Circus* State-rich *RingBuffer*

```
$maxbuff : \mathbb{N}_1$
$maxring = maxbuff - 1$
$Value = \mathbb{N}$
$CellId = 1 \mathinner{\ldotp\ldotp} maxring$
$Direction ::= req \mid ack$

$\mathbf{channel}\ input, output : Value$
$\mathbf{channel}\ write, read, rd\_i, wrt\_i : CellId \times Direction \times Value$
$\mathbf{channel}\ rrd, wrt : Direction \times Value$
$\mathbf{chanset}\ Ctr_I = \{\!| rd\_i, wrt\_i |\!\}$

$\mathbf{process}\ RingCell = \mathbf{begin\ state}\ CellState \mathrel{\widehat=} [v : Value]$
    $InitCell \mathrel{\widehat=} \sqcap x : Value \bullet setV(x)$
    $setV \mathrel{\widehat=} [\Delta CellState;\ x? : \mathbb{N} \mid v' = x?]$
    $Cell = wrt.req?x \rightarrow setV(x);\ wrt.ack.x \rightarrow Skip$
            $\Box\ rrd.req?dumb \rightarrow rrd.ack!v \rightarrow Skip$
    $\bullet\ InitCell;\ (\mu X \bullet Cell;\ X)$
$\mathbf{end}$
$IRCell(i) = RingCell[rrd, wrt := rd\_i.i, wrt\_i.i]$
$DRing = \mathop{|||}\ i : CellId \bullet IRCell(i)$

$\mathbf{process}\ Controller =$
$\mathbf{begin\ state}\ CtrState \mathrel{\widehat=} [cache : Value;\ size : \mathbb{N};\ top : CellId;\ bot : CellId]$
    $InitCtr \mathrel{\widehat=} [CtrState' \mid cache' = 0 \land size' = 0 \land top' = 1 \land bot' = 0]$
    $Input \mathrel{\widehat=}$
        $(size < maxbuff)\ \&$
            $input?x \rightarrow (size = 0)\ \&\ cache := x;\ size := 1$
                        $\Box\ (size > 0)\ \&$
                            $write.top.req!x \rightarrow write.top.ack?dumb \rightarrow$
                            $size := size + 1;\ top := (top \bmod maxring) + 1$
    $Output \mathrel{\widehat=}$
        $(size > 0)\ \&$
            $output!cache \rightarrow$
                $(size > 1)\ \&\ (\sqcap dumb : Value \bullet$
                                    $read.bot.req.dumb \rightarrow read.bot.ack?x \rightarrow Skip);$
                            $size := size - 1;\ bot := (bot \bmod maxring) + 1$
                $\Box\ (size = 1)\ \&\ size := 0$
    $\bullet\ InitCtr;\ \mu X \bullet ((Input \Box Output);\ X)$
$\mathbf{end}$
$ControllerR \mathrel{\widehat=} Controller[read, write := rd\_i, wrt\_i]$
$DBuffer \mathrel{\widehat=} (ControllerR \mathbin{[\![} Ctr_I \mathbin{]\!]} DRing) \setminus Ctr_I$
```

D.3 *Circus* Stateless *RingBuffer*

$maxbuff : \mathbb{N}_1$
$maxring = maxbuff - 1$
$Value = \mathbb{N}$
$CellId = 1 \mathrel{..} maxring$
$Direction ::= req \mid ack$

$\mathbf{channel}\ input, output : Value$
$\mathbf{channel}\ write, read, rd\_i, wrt\_i : CellId \times Direction \times Value$
$\mathbf{channel}\ rrd, wrt : Direction \times Value$
$\mathbf{chanset}\ Ctr_I = \{\!|\ rd\_i, wrt\_i\ |\!\}$

$NAME ::= v \mid top \mid bot \mid cache \mid size$
$BINDING \mathrel{\widehat=} NAME \to \mathbb{U}$
$\delta = \{v \mapsto Value, top \mapsto CellId, bot \mapsto CellId, cache \mapsto Value, size \mapsto \mathbb{N}\}$

$\mathbf{channel}\ mget, mset : NAME \times \mathbb{U}$
$\mathbf{channel}\ terminate$
$MEM_I \mathrel{\widehat=} \{\!|\ mset, mget, terminate\ |\!\}$

$$
\begin{array}{l}
\mathbf{process}\ RingCell = \\
\mathbf{begin} \\
\quad Memory \mathrel{\widehat=} \\
\qquad \mathbf{vres}\ b : BINDING \bullet \\
\qquad\quad (\Box\ n : \operatorname{dom} b \bullet mget.n!b(n) \to Memory(b)) \\
\qquad\quad \Box\ (\Box\ n : \operatorname{dom} b \bullet mset.n?nv : (nv \in \delta(n)) \to Memory(b \oplus \{n \mapsto nv\})) \\
\qquad\quad \Box\ terminate \to Skip \\
\quad \bullet\ \mathbf{var}\ b : \{x : BINDING \mid v \in Value\} \bullet \\
\qquad \left(\begin{array}{l}
\left(\begin{array}{l}
(\sqcap\ v : Value \bullet mget.v?vv : (\delta(v)) \to mset.v!vv \to Skip); \\
\left(\mu X \bullet \left(\begin{array}{l}
mget.v?vv : (\delta(v)) \to \\
\left(\begin{array}{l}
rd.req?dumb \to rd.ack!v \to Skip \\
\Box\ wrt.req?x \to mset.v!x \to \\
\qquad wrt.ack?dumb \to Skip
\end{array}\right)
\end{array}\right); X\right)
\end{array}\right) \\
[\![\emptyset \mid MEM_I \mid \{b\}]\!] \\
Memory(b)
\end{array}\right) \\
\qquad \setminus MEM_I \\
\mathbf{end}
\end{array}
$$

$IRCell(i) = RingCell[rrd, wrt := rd\_i.i, wrt\_i.i]$
$DRing = |\!|\!|\ i : CellId \bullet IRCell(i)$

```
process $Controller$ =
begin
    $Memory \mathrel{\widehat=}$
        vres $b : BINDING \bullet$
            $(\Box\ n : \mathrm{dom}\ b \bullet mget.n!b(n) \rightarrow Memory(b))$
            $\Box\ (\Box\ n : \mathrm{dom}\ b \bullet mset.n?nv : (nv \in \delta(n)) \rightarrow Memory(b \oplus \{n \mapsto nv\}))$
            $\Box\ terminate \rightarrow Skip$
    $\bullet$ var $b : \left\{ \begin{array}{l} x : BINDING \mid cache \in Value \land size \in \mathbb{N} \\ \qquad \land\ top \in CellId \land bot \in CellId \end{array} \right\} \bullet$
        (
          (
            $mget.cache?vcache : (\delta(cache)) \rightarrow mget.size?vsize : (\delta(size)) \rightarrow$
            $mget.top?vtop : (\delta(top)) \rightarrow mget.bot?vbot : (\delta(bot)) \rightarrow$
            $mset.cache.0 \rightarrow mset.size.0 \rightarrow mset.top.1 \rightarrow mset.bot.1 \rightarrow$
            $\mu X \bullet$
                (
                  $mget.cache?vcache : (\delta(cache)) \rightarrow$
                  $mget.size?vsize : (\delta(size)) \rightarrow$
                  $mget.top?vtop : (\delta(top)) \rightarrow$
                  $mget.bot?vbot : (\delta(bot)) \rightarrow$
                  $(vsize < maxbuff)\ \&$
                      $input?x \rightarrow$
                          $(vsize = 0)\ \&$
                              $mset.cache.x \rightarrow mset.size.1 \rightarrow Skip$
                          $\Box\ (vsize > 0)\ \&$
                              $write.vtop.req!x \rightarrow$
                              $write.vtop.ack?dumb \rightarrow$
                              $mset.size.(vsize + 1) \rightarrow$
                              $mset.top.(vtop\ \mathsf{mod}\ vmaxring) \rightarrow$
                              $Skip$
                  $\Box\ (vsize > 0)\ \&$
                      $output!cache \rightarrow$
                          $(vsize > 1)\ \&$
                              ( $\sqcap\ dumb : Value \bullet$
                                    $read.vbot.req.dumb \rightarrow$
                                    $read.vbot.ack?x \rightarrow$
                                    $mset.cache.x \rightarrow Skip$ ) ;
                              $mset.size.(vsize - 1) \rightarrow$
                              $mset.bot.((vbot\ \mathsf{mod}\ maxring) + 1) \rightarrow$
                              $Skip$
                          $\Box\ (vsize = 1)\ \&\ mset.size.0 \rightarrow Skip$
                ) ;
            $X$
          )
          $[\![\emptyset \mid MEM_I \mid \{b\}]\!]$
          $Memory(b)$
        )
        $\setminus MEM_I$
end
```

$$\begin{aligned}
ControllerR &\mathrel{\widehat=} Controller[read, write := rd\_i, wrt\_i] \\
DBuffer &\mathrel{\widehat=} (ControllerR \mathbin{[\![} Ctr_I \mathbin{]\!]} DRing) \setminus Ctr_I
\end{aligned}$$

D.4 CSP *RingBuffer*

```
--------------------------------------------------------------------
--------------------------------------------------------------------
-- Function auxiliary operations
--------------------------------------------------------------------
--------------------------------------------------------------------
-- Function is a set {(x1, y1),...,(xn, yn)}

-- Transforms a singleton set into the element itself
pick({x}) = x

-- Returns the function return
-- Raises error if z is not in the domain of the function
apply(f,x) = pick({ v | (n,v) <- f, n==x})

-- domain antirestriction
ddres(f,xs) = {(n,v) | (n,v) <- f, not member(n,xs)}

-- domain restriction
dres(f,xs) = {(n,v) | (n,v) <- f, member(n,xs)}

-- Overwrites the fuction
over(f,n,v) = union(ddres(f,{n}),{(n,v)})

-- Returns the domain of a relation
dom(f) = {n | (n,v) <- f}

-- Returns the domain of a relation
ran(f) = {v | (n,v) <- f}

--------------------------------------------------------------------
--------------------------------------------------------------------
-- Sequence auxiliary operations
--------------------------------------------------------------------
--------------------------------------------------------------------
-- Overriding
```

```
insert(<>,i,x) = <>
insert(<y>^s,n,x) = if (n==1) then <x>^s else <y>^insert(s,n-1,x)

--------------------------------------------------------------------
-- Indexing
at(<x>^s,n) = if (n==1) then x else at(s,n-1)

--------------------------------------------------------------------
-- Sequence of 0s
zeroSeq(n) = if (n==1) then <0> else <0>^zeroSeq(n-1)
--------------------------------------------------------------------

map(f,<>) = <>
map(f,<x>^xs) = <f(x)>^(map(f,xs))

e2l(x) = <x>

addHead(e,<>) = <>
addHead(e,<x>^xs) = <<e>^x>^(addHead(e,xs))

seqCons(<>,xxs) = <>
seqCons(<x>^xs,xss) = (addHead(x,xss))^(seqCons(xs,xss))

distCartProd(<>) = <>
distCartProd(<xs>) = map(e2l,xs)
distCartProd(<xs>^xxs) = seqCons(xs, distCartProd(xxs))


--------------------------------
-- GENERAL DEFINITIONS
--------------------------------

-- The maximum size of the buffer is a strictly positive constant.
maxbuff = 3

-- The values buffered are numbers.
Value = {0..2}

-- The ring is a circular array, modelled as a sequence whose two
-- ends are considered to be joined.
-- The constant maxring, defined as (maxbuff - 1), gives the bound for
-- the ring.
maxring = maxbuff - 1

-- The communication is bi-directional
datatype Direction = req | ack

CellId = {1 .. maxring}

--------------------------------
-- THE ABSTRACT BUFFER
--------------------------------

-- It takes its inputs and supplies its outputs on two different
-- typed channels.
channel input, output: Value

ABuffer =
  let BufferState(s)= #s > 0 & output!head(s) -> BufferState(tail(s))
                      []
                      #s < maxbuff & input?x -> BufferState(s ^ <x>)

  within
    BufferState(<>)


--------------------------------
-- BINDINGS
--------------------------------

--------------------------------
-- Set of names
datatype NAME = RingCell_v | Controller_top | Controller_bot
                | Controller_cache | Controller_size

--------------------------------
-- Nat
NatValue = {0..maxbuff}

--------------------------------
-- The universe of values
datatype UNIVERSE = Boolean.Bool | Nat.NatValue | Val.Value | Cel.CellId
```

```
--------------------------------
-- Conversions
subtype U_BOOL = Boolean.Bool
subtype U_NAT = Nat.NatValue
subtype U_VALUE = Val.Value
subtype U_CELL = Cel.CellId

value(Nat.v) = v
value(Boolean.v) = v
value(Val.v) = v
value(Cel.v) = v

type(x) =
    if x== RingCell_v then U_VALUE
    else if x == Controller_top then U_CELL
    else if x == Controller_bot then U_CELL
    else if x == Controller_cache then U_VALUE
    else if x == Controller_size then U_NAT
    else {}

tag(x) =
    if x== RingCell_v then Val
    else if x == Controller_top then Cel
    else if x == Controller_bot then Cel
    else if x == Controller_cache then Val
    else if x == Controller_size then Nat
    else Nat

--------------------------------
-- All possible bidings
NAMES_VALUES = seq({seq({(n,v) | v <- type(n)}) | n <- NAME})
BINDINGS = {set(b) | b <- set(distCartProd(NAMES_VALUES))}

--------------------------------
-- MEMORY
--------------------------------
channel mget, mset : NAME.UNIVERSE
channel terminate

MEM_I = {| mset, mget, terminate |}
```

```
Memory(b) =
    ([] n:dom(b) @ mget.n!(apply(b,n)) -> Memory(b))
    [] ([] n:dom(b) @ mset.n?x:type(n) -> Memory(over(b,n,x)))
    [] terminate -> SKIP

Memorise(P, b) =
    ((P; terminate -> SKIP) [| MEM_I |] Memory(b)) \ MEM_I

--------------------------------
-- STATELESS RING
--------------------------------
channel rd, wrt: Direction . Value

RingCellMain =
    (|~| v:Value @
        mget.RingCell_v?vRingCell_v:(type(RingCell_v)) ->
        mset.RingCell_v!((tag(RingCell_v)).v) ->
        SKIP);
    (let
        MuCellX =
            (mget.RingCell_v?vRingCell_v:(type(RingCell_v)) ->
                (
                rd.req?dumb ->
                    rd.ack!(value(vRingCell_v)) ->
                    SKIP
                []
                wrt.req?x ->
                    mset.RingCell_v!((tag(RingCell_v)).x) ->
                    wrt.ack?dumb ->
                    SKIP
            ));
            MuCellX
    within
        MuCellX)

MemoryRingCell =
    let restrict(bs) = dres(bs,{RingCell_v})
    within
        |~| b:BINDINGS @ Memorise(RingCellMain, restrict(b))
```

```
-- An indexed cell
channel rd_i, wrt_i: CellId . Direction . Value

MemoryIRCell(i) = MemoryRingCell [[rd <- rd_i.i, wrt <- wrt_i.i]]

-- The distributed ring
MemoryDRing = ||| i: CellId @ MemoryIRCell(i)

--------------------------------
-- STATELESS CONTROLLER
--------------------------------
channel write, read: CellId . Direction . Value

ControllerMain =
    (mget.Controller_cache?vController_cache:(type(Controller_cache)) ->
     mget.Controller_size?vController_size:(type(Controller_size)) ->
     mget.Controller_top?vController_top:(type(Controller_top)) ->
     mget.Controller_bot?vController_bot:(type(Controller_bot)) ->
     mset.Controller_cache.((tag(Controller_cache)).0) ->
     mset.Controller_size.((tag(Controller_size)).0) ->
     mset.Controller_top.((tag(Controller_top)).1) ->
     mset.Controller_bot.((tag(Controller_bot)).1) ->
     SKIP);
let
    MuControllerX =
        (
        mget.Controller_cache?vController_cache:(type(Controller_cache)) ->
        mget.Controller_size?vController_size:(type(Controller_size)) ->
        mget.Controller_top?vController_top:(type(Controller_top)) ->
        mget.Controller_bot?vController_bot:(type(Controller_bot)) ->
        (
           (value(vController_size) < maxbuff) &
                input?x ->
                (
                (value(vController_size) == 0) &
                    mset.Controller_cache.((tag(Controller_cache)).x) ->
                    mset.Controller_size.((tag(Controller_size)).1) ->
                    SKIP
                []
                (value(vController_size) > 0) &
                    write.(value(vController_top)).req!x ->
                    write.(value(vController_top)).ack?dumb ->
                    mset.Controller_size.((tag(Controller_size))
                                        .((value(vController_size))+1)) ->
                    mset.Controller_top.((tag(Controller_top))
                                       .(((value(vController_top))
                                           % maxring)+1)) ->
                    SKIP
                )
           []
           (value(vController_size) > 0) &
                output!(value(vController_cache)) ->
                (
                (value(vController_size) > 1) &
                    (|~| dumb:Value @
                        read.(value(vController_bot)).req.dumb ->
                        read.(value(vController_bot)).ack?x ->
                        mset.Controller_cache.((tag(Controller_cache)).x) ->
                        SKIP);
                    (mset.Controller_size.((tag(Controller_size))
                                         .((value(vController_size))-1)) ->
                     mset.Controller_bot.((tag(Controller_bot))
                                        .(((value(vController_bot))
                                            % maxring)+1)) ->
                     SKIP)
                []
                (value(vController_size) == 1) &
                        mset.Controller_size.((tag(Controller_size)).0) ->
                        SKIP
                )
            )
        );
        MuControllerX
within
    MuControllerX

MemoryController =
    let restrict(bs) = dres(bs,{Controller_cache, Controller_size,
                                Controller_top, Controller_bot})
    within
        |~| b:BINDINGS @ Memorise(ControllerMain, restrict(b))
```

```
--------------------------------
-- THE RING BUFFER
--------------------------------
MemoryControllerR = MemoryController[[ read <- rd_i, write <- wrt_i]]
MemoryDBuffer =
    (MemoryControllerR [| {| rd_i, wrt_i |} |] MemoryDRing)
    \ {| rd_i, wrt_i |}
```

E Lifting the Approach to Circus and CML

In this section, we present some important definitions that are referenced elsewhere in this document.

Definition E.1 (Naive client-server component) *Let Ctr be a component contract. Then Ctr is a naive client-server component if, and only if:* $\forall c, P \mid c \in \mathcal{C}_{Ctr} \land P = Prot_{IMP}(Ctr, c) \bullet StrictProt(P, c, in, out) \lor StrictProt(P, c, out, in)$ *where*

$$\begin{aligned} StrictProt(P, c, d_1, d_2) = \forall s : traces(P) \bullet (s \downarrow \{c.d_1\} \geq s \downarrow \{c.d_2\}) \land \\ (s \downarrow \{c.d_1\} \leq s \downarrow \{c.d_2\} + 1) \end{aligned}$$

Definition E.2 (Communication protocol) *We say a CSP process P is a communication protocol if :*

- $\exists c_1, c_2 \bullet inputs(P) \subseteq \{c_1\} \land outputs(P) \subseteq \{c_2\}$;

Definition E.3 (Dual protocol) *Let P be a deadlock-free communication protocol. The dual protocol of P is defined as a deadlock-free communication protocol DP such that:*

$$\begin{aligned} & inputs(P) = outputs(DP) \\ \land\ & outputs(P) = inputs(DP) \\ \land\ & traces(DP) = traces(P) \end{aligned}$$

E.1 Propositions

Proposition E.1 (Renaming and I/O Processes) *Let P be an I/O process, c and z I/O channels, and R a bijection from all input and outputs events of P in c into events of z. Then,* $P \llbracket R \rrbracket$ *is also an I/O process.*

Since R is a bijection, there is strong bisimulation relation among states of P and $P \llbracket R \rrbracket$. Furthermore, all properties directly related to the traces and failures of P, are also valid to $P \llbracket R \rrbracket$. Moreover, the channel z replaces c in all properties that takes the I/O channels used by P. As a consequence, $P \llbracket R \rrbracket$ satisfy all properties that P satisfies to be an I/O process. This is an important proposition that underpins the notions of component instantiation and protocol equivalences. Based on this, the result of proved properties about an I/O process (or protocol) can also be applied to a renamed version of it, which satisfy the statement in this proposition. Similarly, more elaborated observations about a component can be applied to renaming versions of it, like the property of a component belongs to an client-server style architecture.

E.2 Theorems

Theorem E.1 *(Protocol Implementation and Deadlock freedom)*
If P is deadlock-free, then $Prot_{IMP}(P, ic)$ is also deadlock-free, for any ic and oc.

Proof. If P is deadlock free, there is no trace tr such that $(tr, \alpha P)$ is in $failures(P)$; the failures of the projection of P over channel ic is a subset of $failures(P)$; $Prot_{IMP}(P, ic)$ is a failures-divergence refinement of the projection of P over channel ic; hence, its failures is a subset of the failures of the projection; finally, if in a bigger set of failures there is no trace such that $(tr, \alpha P)$ is in $failures(P)$, the smaller set also has this property.

Theorem E.2 *(Divergence freedom)*
If a process P has no hiding and no unguarded recursion then P is divergence-free.

Proof: This theorem is straightforward result of the semantic calculation of the standard CSP operators in [Ros98]. According to him, these are the operators that cause divergences.

Theorem E.3 *(Client-Server Architectures Properties)*
If Q is a client-server protocol, then it satisfies the finite output property and is I/O Confluent, for any ic and oc.

Proof. We prove this theorem in two parts. The first part is dedicated to prove that Q satisfies the Finite Output Property, and the second one to prove that Q is I/O confluent.

According to Definition E.2, a protocol is a process that communicates all inputs through a unique channel, as well as, all outputs through a unique channel. These can be two distinct channels, or same channel to communicate inputs and outputs. However, in order to be a naïve client-server, the communications via a channel must follow a strict pattern of communications of inputs and outputs. So, all inputs and outputs of Q are performed via a same channel.

1. As Q is naïve client-server and all inputs and outputs are communicated via a same channel in a strict pattern - in which inputs and outputs must be interspersed - after an output being communicated, an input is mandatory. As a consequence, we conclude that the Finite Output Property is satisfied.
2. According to Definition E.1, and as explained above, a naïve client-server process cannot perform two subsequent outputs, neither it can perform subsequent inputs. So, it does not present a choice among inputs and outputs. Moreover, as already explained, inputs and outputs are communicated through a same channel. As a consequence, we can state following, which implies in I/O confluence.

Theorem E.4 *(Renaming and Dual Protocols Distribution)*
$Prot_{DUAL}((Prot_{IMP}(P, ic)) \llbracket R_{IO}^{ic \to oc} \rrbracket) = Prot_{DUAL}(Prot_{IMP}(P, ic)) \llbracket R_{IO}^{ic \to oc} \rrbracket$

Proof. As the properties of a process being a protocol is closed over renaming (Proposition E.1), we only have to proof the following three things, according to Definition E.3. To easy the reading, consider $Q = Prot_{IMP}(P, ic)$, $R' = R_{IO}^{ic \to oc}$.

- $inputs(Q \llbracket R' \rrbracket) = outputs(Prot_{DUAL}(Q) \llbracket R' \rrbracket)$

$$\begin{aligned}
& outputs(Prot_{DUAL}(Q) \llbracket R' \rrbracket) \\
&= \{e \mid e \in outputs(Prot_{DUAL}(Q) \llbracket R' \rrbracket)\} \\
&= \{R'(e) \mid e \in outputs(Prot_{DUAL}(Q))\} \\
&= \{R'(e) \mid e \in inputs(Q)\} \\
&= \{e \mid e \in inputs(Q \llbracket R' \rrbracket)\} \\
&= inputs(Q \llbracket R' \rrbracket)
\end{aligned}$$

- $outputs(Q \llbracket R' \rrbracket) = inputs(Prot_{DUAL}(Q) \llbracket R' \rrbracket)$

  Similar to the proof of $inputs(Q \llbracket R' \rrbracket)$

- $traces(Q \parallel [\![R']\!]) = traces(Prot_{DUAL}(Q) \parallel [\![R']\!])$

$$\begin{aligned}
& traces(Prot_{DUAL}(Q) \parallel [\![R']\!]) \\
& = \{ e \mid e \in traces(Prot_{DUAL}(Q) \parallel [\![R']\!]) \} \\
& = \{ R'(e) \mid e \in traces(Prot_{DUAL}(Q)) \} \\
& = \{ R'(e) \mid e \in traces(Q) \} \\
& = \{ e \mid e \in traces(Q \parallel [\![R']\!]) \} \\
& = traces(Q \parallel [\![R']\!])
\end{aligned}$$

Theorem E.5 *(Protocol Implementation Instantiation for Forks)*
$Prot_{FK}(c)$ is a valid protocol implementation of any instantiation of FORK that renames either $fk1$ or $fk2$ to c.

Proof. Direct result of Proposition E.1.

Theorem E.6 *(Protocol Implementation Instantiation for Philosophers)*
$Prot_{PH}(c)$ is a valid protocol implementation of any instantiation of PHIL that renames either $pfk1$ or $pfk2$ to c.

Proof. Direct result of Proposition E.1.

Theorem E.7 *(Finite Buffers and Finite Output Property)*
The composition operator considers the existence of infinite buffers in the medium. In case, we know the medium can be specified as a buffer of finite size, satisfying the finite output property is immaterial.

Proof. From [Ram11].

F Mechanisation of the Composition Rules Side Conditions in CSP

In this section we list the CSP assertions that mechanise the side conditions of the composition rules.

F.1 Interleave composition ($P\ [|||]\ Q$)

A.1 Alphabets are disjoint
`assert STOP [T= RUN(inter(events(P),events(Q)))`[2]

[2]Assertions painted in red can be solved by SAT solvers (Removed at Level 2)

A.2 P is an I/O Process

A.2.1 : Every channel in P is an I/O Channel

```
assert not Test(inter(inputs(P),outputs(P)) == {})
             [T= ERROR
```

A.2.2 : P has infinite traces

`assert not HideAll(P):[divergence free [FD]]`[3]

A.2.3 : P is divergence-free

`assert P:[divergence free [FD]]`[4]

A.2.4 : P is input deterministic

```
assert LHS_InputDet(P) [F= RHS_InputDet(P)
```

A.2.5 : P is strong output decisive

```
assert LHS_OutputDec_A(P) [F= RHS_OutputDec_A(P)
assert LHS_OutputDec_B(P,c1) [F= RHS_OutputDec_B(P,c1)
...
assert LHS_OutputDec_B(P,cn) [F= RHS_OutputDec_B(P,cn)
```

A.3 : Q is an I/O Process
Similar to A.2

F.2 Communication composition ($P[ip \leftrightarrow oq]Q$)

D.0 : Both are an I/O Process

D.0.1 : P is an I/O Process
Similar to A.2

D.0.2 : Q is an I/O Process
Similar to A.2

D.1 : ip is in the alphabet of P

```
assert not P \ {| ip |} [T= P
```

D.2 : iq is in the alphabet of Q
Similar to D.1

[3]Assertions painted in orange may be discarded if they are applied to components resulting from previous compositions. They are achieved by composition using Theorems 4.1 to 4.4 from [RSM09] (Removed at Level 2)

[4]Assertions painted in blue may be discarded using syntactic restrictions based on Theorem E.2 (Removed at Level 2)

D.3 : Alphabets are disjoint
Similar to A.1

D.4 : $Prot_{IMP}(P, ip) \llbracket R_{IO}^{ip \to iq} \rrbracket$ is I/O Confluent

– Finding a valid protocol implementation

D.4.1 : It is divergence-free

```
assert apply(PROT_IMP, (P,ip)) :[divergence free [FD]]
```

D.4.2 : It is refined by the projection on the channel

```
assert apply(PROT_IMP, (P,ip))[F= PROT_IMP_def(P,ip)
```
[5]

D.4.3 : It is a refinement of the projection on the channel

```
assert PROT_IMP_def(P,ip) [FD= apply(PROT_IMP, (P,ip))
```

D.4.4 : It is a port-protocol (communication protocol)

D.4.4.1 : inputs

```
assert not Test(subseteq(apply(inputs_PROT_IMP,(P,ip)),
                                     {| fk |}))
                [T= ERROR
```

D.4.4.2 : outputs

```
assert not Test(subseteq(apply(outputs_PROT_IMP,(P,ip)),
                                     {| fk |}))
                [T= ERROR
```

D.4.5 : The renamed version is I/O Confluent

```
assert InBufferProt(PROT_IMP_R(P,ip, RP), ip)
       :[deterministic [F]]
```

D.5 : $Prot_{IMP}(P, iq) \llbracket R_{IO}^{iq \to ip} \rrbracket$ is I/O Confluent
Similar to D.4

D.6 : Protocols are Strong Compatible

D.6.1 : Protocols are deadlock-free

D.6.1.1 : Left

```
assert PROT_IMP_R(P,ip,RP) :[deadlock free [FD]]
```
[6]

D.6.1.2 : Right

```
assert PROT_IMP_R(Q,iq,RQ) :[deadlock free [FD]]
```

[5]Assertions painted in dark green may be discarded by using metadata to calculate protocol implementations and dual protocols (Removed at Level 4)

[6]Assertions painted in magenta may be discarded if P (and Q) is a result from previous compositions based on Theorems 3.1 and E.1 (Removed at Level 2)

D.6.2 : Protocols are communication protocols

D.6.2.1 : Left inputs

```
assert not Test(subseteq(inputs_PROT_IMP_R(P,ip,R),
                             {| fk |})) [T= ERROR
```

D.6.2.2 : Left outputs

```
assert not Test(subseteq(outputs_PROT_IMP_R(P,ip,R),
                             {| pfk |}))[T= ERROR
```

D.6.2.3 : Right Inputs

```
assert not Test(subseteq(inputs_PROT_IMP_R(Q,iq,RQ),
                             {| pfk |}))[T= ERROR
```

D.6.2.4 : Right Outputs

```
assert not Test(subseteq(outputs_PROT_IMP_R(Q,iq,RQ),
                             {| fk |}))[T= ERROR
```

D.6.3 : We have a Dual Protocol

D.6.3.1 : inputs and outputs

```
assert not Test(outputs_DUAL_PROT_IMP_R(P,ip,DUAL_R)
                  ==  inputs_PROT_IMP_R(P,ip,R))[T= ERROR
```

D.6.3.2 : inputs and outputs

```
assert not Test(inputs_DUAL_PROT_IMP_R(P,ip,DUAL_R)
                  == outputs_PROT_IMP_R(P,ip,R))[T= ERROR
```

D.6.3.3 : are trace equivalent

D.6.3.3.1 : Left

```
assert DUAL_PROT_IMP_R(P,ip,DUAL_RP)
       [T= PROT_IMP_R(P,ip,RP)
```

D.6.3.3.2 : Right

```
assert PROT_IMP_R(P,ip,RP)
       [T= DUAL_PROT_IMP_R(P,ip,DUAL_RP)
```

D.6.4 `assert DUAL_PROT_IMP_R(P,ip,DUAL_RP)`
`[F= PROT_IMP_R(Q,iq,RQ)`

D.6.5 : Matching Compatibility

```
assert DUAL_PROT_IMP_R(P,ip,DUAL_RP)[7]
       [F= PROT_IMP_R(Q,iq,RQ)
```

D.7 : Protocols have Finite Output Property

[7] Assertions marked in apricot should be included only at Levels 4 and 5

D.7.1 : Left

```
assert PROT_IMP_R(P,ip,R) \ allOutputs[8]
       :[divergence free [FD]]
```

D.7.2 : Right

```
assert PROT_IMP_R(Q,iq,RQ) \ allOutputs
       :[divergence free [FD]]
```

F.3 Feedback composition ($P[ip \hookrightarrow oq]$)

E.0 : P is an I/O Process
Similar to D.0

E.1 : *ip* is in the alphabet of P
Similar to D.1

E.2 : *oq* is in the alphabet of Q
Similar to D.1

E.3 : $Prot_{IMP}(P, ip) \parallel R_{IO}^{ip \rightarrow oq} \|$ is I/O Confluent
Similar to D.4

E.4 : $Prot_{IMP}(P, oq) \parallel R_{IO}^{oq \rightarrow ip} \|$ is I/O Confluent
Similar to D.4

E.5 : Protocols are Strong Compatible
Similar to D.6

E.6 : Protocols have Finite Output Property
Similar to D.7

E.7 : Channels *ip* and *oq* are decoupled in P

E.7.1 Left

```
assert INTER_PROT_IMP(P,{ip, oq})[9]
       [F= PROJECTION(P,{ip, oq})
```

E.7.2 Right

```
assert PROJECTION(P,{ip, oq})
       [FD= INTER_PROT_IMP(P,{ip, oq})
```

[8]Assertions painted in purple may be discarded if we are using finite buffers based on Theorem E.7 (Removed at Level 3)

[9]Assertions painted in brown may be discarded by using metadata to calculate decoupled channels (Removed at Level 4)

F.4 Reflexive composition ($P[ip \hookrightarrow op]$)

I.1 : *ip* is in the alphabet of P
Similar to D.1

I.2 : *op* is in the alphabet of Q
Similar to D.1

I.3 : $P \upharpoonright \{ip, op\}$ is buffering self-injection compatible

I.3.1 : P is deadlock-free

```
assert P :[deadlock free [FD]]
```

I.3.2 : P is an I/O Process
Similar to A.2

I.3.3 $Prot_{IMP}(P, ip) \llbracket LR1 \rrbracket$ and $Prot_{IMP}(P, op) \llbracket LR2 \rrbracket$ are strong compatible

I.3.3.1-2 Finding a valid protocol implementation for P on *ip*: Similar do D.4.1 to D.4.3

I.3.3.3-4 Finding a valid protocol implementation for P on *op*: Similar do D.4.1 to D.4.3

I.3.3.5 Protocols are communication protocols: Similar do D.6.2 replacing Q by P

I.3.3.6 : Protocols (with renaming) are Strong Compatible
Similar to D.6

I.3.3.7 : $Prot_{IMP}(P, ip) \llbracket LR1 \rrbracket$ and $Prot_{IMP}(P, op) \llbracket LR2 \rrbracket$ have finite output property Similar to D.7

I.3.3.8 P in parallel with the $BUFFER_{IO}$ using the renaming is deadlock-free

```
assert not PROJECTION(P,{pi, po})
           [| {| pi, po |} |]
           BUFF_IO_1(P_LR1, P_LR2) :[deadlock free [F]]
```

G An Exercise on the New Definition of Channel Projection

In this section, we present an exercise on the new definition of channel projection. All the discussion is based on CSP, since the theoretical background of the overall approach is also CSP based.

In this section we use the following examples:

```
P1(x,y) =
    c1.out.x -> c3.in -> P1(x,y)
    []
    c2.in -> P1(x,y)
    []
    c1.in.x ->
        (
        c1.out.y -> (P1(x,y) |~| P1(not x, not y))
        |~|
        c3.out -> P1(x,y)
        )

Prot(x,y) =
    c1.out.x -> Prot(x,y)
    []
    c1.in.x ->
        (
        c1.out.y -> (P1(x,y) |~| P1(not x, not y))
        |~|
        Prot(x,y)
        )
```

This gives us a protocol `Prot(x,y)` that satisfies the conditions under which a projection of `P1` on `c1` is valid.

We need to redefine projection $P \upharpoonright C$ in order to remove the hiding, which is currently defined as follows:

$$P \upharpoonright C \mathrel{\widehat{=}} P \setminus (\Sigma \setminus \bigcup_{c:C} \{\!| \ c \ |\!\})$$

There are two options for that:

1. using lazy abstraction as proposed by Bill Roscoe in [Ros98], or;

2. Find another way to express using the traces model together with I/O Process properties

In what follows, we present a discussion on both approaches.

G.1 Lazy Abstraction

Every process will have some environment interacting with it on the events you are not concentrating on. We need to build a model of what that interaction looks like:

- Hiding assumes that the other events are always available: the environment always offers them to the process.
- Lazy abstraction implies that the environment might always accept or refuse any event.
- Mixed abstraction partitions them into events that the environment can refuse and ones it can't.

Using lazy abstraction, we might use one of the three proposals in [Ros98]. However, the option on the traces model ($\mathcal{T}$) cannot be expressed in FDR. We are left with the infinite traces model ($\mathcal{U}$) and the failures model ($\mathcal{F}$).

G.1.1 Lazy Abstraction in the Failures Model ($\mathcal{F}$).

The proposal for the failures model is defined as follows:

```
LAZY_F(L,H,P) = (P [| H |] CHAOS(H)) \ H

assert P1(true,false) \ {| c2, c3 |}
        [FD= LAZY_F({| c1 |},{| c2, c3 |}, P1(true,false))
assert LAZY_F({| c1 |},{| c2, c3 |},P1(true,false))
        [F= P1(true,false) \ {| c2, c3 |}
assert LAZY_F({| c1 |},{| c2, c3 |} ,P1(true,false))
        :[livelock free]
assert LAZY_F({| c1 |},{| c2, c3 |}, P1(true,false))
        :[deadlock free [FD]]
```

Although it works in the behavioural checking, there are two problems with this approach:

1. It is not divergence free: it might still have an infinite loop with internal (eager) events, that is, events in `H`.
2. It in not deadlock free: `CHAOS` might refuse events that are in `H` causing a deadlock.

Lazy Abstraction in the Infinite Traces Model ($\mathcal{U}$). The infinite traces model proposal has the expected behaviour and is livelock free; however it is not deadlock free.

```
LAZY_U(L,H,P) = (P [| Events |] SemiFair(L,H)) \ H

assert P1(true,false) \ {| c2, c3 |}
        [FD= LAZY_U({| c1 |},{| c2, c3 |},P1(true,false))
assert LAZY_U({| c1 |},{| c2, c3 |},P1(true,false))
        [F= P1(true,false) \ {| c2, c3 |}
assert LAZY_U({| c1 |},{| c2, c3 |},P1(true,false))
        :[livelock free]
assert LAZY_U({| c1 |},{| c2, c3 |},P1(true,false))
        :[deadlock free]
```

The possibility of deadlock comes from the fairness used to avoid infinite internal loops. The approach to establish fairness forces at some point the hidden events not to be offered. That might be the exact point in which the process actually wants to synchronise only on those events causing a deadlock.

Finally, because hiding might introduce divergence and lazy abstraction might introduce deadlock, mixed approaches might introduce both.

In a draft of "Fairness analysis through priority", Bill Roscoe faces the same problem when trying to abstract from a clock event `Me.Clock`; the solution is given using the priority model:

"*In the CSP models, the modelling event Me.Clock cannot be lazily abstracted, since that would imply that it could choose never to happen. On the other hand, simply hiding it means that it is eager and therefore could occur instantaneously. This in turn can cause a divergence in the model exploiting the eager modelling event, which at runtime is not regarded as erroneous behaviour under the assumed circumstances.*
...
We therefore need CSP models such that the benign divergences are ignored, without losing any behaviour, including genuine divergences called malign

divergences, in the design that may result in genuine errors that need to be found. Conventional CSP cannot solve this problem, but a solution is achieved using the priority-based techniques described in Section 8."

In our context, we do not want to move away from the "standard" models $\mathcal{T}$, $\mathcal{F}$, and $\mathcal{FD}$ because we want to reuse previous results from [CG10, CG07] that relates CSP refinement and ***Circus*** refinement. Since, we cannot specify the abstracted process algebraically, we proposed a solution in which we define properties that needs to be satisfied by a candidate abstracted process. The withdraw is that given a process P, we do not have an algebraic definition of what its abstraction is - this would constitute a function $F(P, c)$ that would return such abstraction - instead, the user will propose a candidate that will need to satisfy these properties.

G.2 Traces Model and I/O Process Properties

We propose the projection plays a role simply in the traces of a process. Hence, we would have that the projection on c of process P is a process that:

- Does not have any event other than c
- Has exactly the same traces as P on c; the behaviour on the other events are irrelevant.

For that, instead of calculating a given projection, the user of the strategy needs to propose a projection that satisfies these properties. This, however, might be automated by a syntactic function that removes the channel. Nevertheless, we also need to guarantee that the communication directions (input and output) are not changed and that the properties of strong output decisiveness and input determinism are maintained. Overall, these properties would be characterised as follows:

New Definition 4.3 (Projection) *Let P be an I/O Process, and C a set of communication channels. The projection of P over C (denoted by $P \upharpoonright C$) satisfies the following properties:*

1. *$P \upharpoonright C$ is an I/O Process*
2. $\forall c : C \bullet inputs(P \upharpoonright C, c) \subseteq inputs(P, c)$
3. $\forall c : C \bullet outputs(P \upharpoonright C, c) \subseteq outputs(P, c)$

4. $\alpha(P \upharpoonright C) \subseteq \bigcup_{c:C} \{\!| c |\!\}$
   - The new CSP test characterisation is checking that $\mathit{ProtCheck}(P \upharpoonright C, C)$ is deadlock-free, where:

$$\mathit{ProtCheck}(P, C) = P \,[\![\, \mathit{NOT}(C) \,]\!]\, \mathit{PRUNE}(\mathit{NOT}(C))$$
$$\mathit{PRUNE}(A) = \Box\, ev : A \bullet ev \to \mathit{Stop}$$
$$\mathit{NOT}(C) = \Sigma \setminus \bigcup_{c:C} \{\!| c |\!\}$$

5. $P \equiv_{\mathrm{T}} P \,[\![\, \Sigma \,]\!]\, ((P \upharpoonright C) \,|||\, \mathit{RUN}(\mathit{NOT}(C)))$

$$\mathit{RUN}(\mathit{CS}) = \Box\, c : \mathit{CS} \bullet c \to \mathit{RUN}(\mathit{CS})$$

Properties 1 - 3 guarantees that the communication direction (input and output) are not changed and that the properties of strong output decisiveness and input determinism are maintained. This ensures that we are neither removing not introducing non-determinism. Property four ensures that the projection process refers only to channels in C. Finally, together with the previous properties, property 5 guarantees that the process behaviour on the projected channels is not changed.

H Z Formalisation of BRIC

In what follows we present a Z type-checked formalisation of the compositional model from [Ram11].

section *csp_circus_toolkit* parents *standard_toolkit*

H.1 Embedding *Circus* Syntax into Z

First, some Z syntax constructs are irrelevant for the definition of presented here. Namely, they are the Z names, declarations, expressions, predicates, schema expressions, and the left-hand side of definitions [Spi92]. Therefore, they are defined as Z given sets.

$[N, \mathit{Pred}, \mathit{SchemaExp}, \mathit{DefLHS}]$

$\mathit{BOOL} ::= \mathit{TRUE} \mid \mathit{FALSE}$

First, some *Circus* syntax constructs are also irrelevant for the definition of this work. They are channel declarations, name set expressions and communications.

$$[\mathit{CDecl}, \mathit{NSExp}, \mathit{CHANNEL}]$$

On the other hand, channel set expressions may be a set display of channel names, a reference to a previously defined channel set, or a composition of two other channel set (union, intersection, or difference).

$$\begin{array}{lll}
\mathit{CSExp} ::= & \mathit{CSDisplay}\langle\langle \mathbb{P}\, \mathit{CHANNEL} \rangle\rangle \\
\quad | & \mathit{CSName}\langle\langle N \rangle\rangle \\
\quad | & \cup\langle\langle \mathit{CSExp} \times \mathit{CSExp} \rangle\rangle \\
\quad | & \cap\langle\langle \mathit{CSExp} \times \mathit{CSExp} \rangle\rangle \\
\quad | & \setminus\langle\langle \mathit{CSExp} \times \mathit{CSExp} \rangle\rangle
\end{array}$$

$$\begin{array}{lll}
\mathit{VALUE} ::= & \mathit{Bool}\langle\langle \mathit{BOOL} \rangle\rangle \\
\quad | & \mathit{Int}\langle\langle \mathbb{Z} \rangle\rangle \\
\quad | & \mathit{Seq}\langle\langle \mathrm{seq}\, \mathit{VALUE} \rangle\rangle \\
\quad | & \mathit{Pair}\langle\langle (\mathit{VALUE} \times \mathit{VALUE}) \rangle\rangle \\
\quad | & \mathit{Set}\langle\langle \mathbb{P}\, \mathit{VALUE} \rangle\rangle \\
\quad | & \mathit{Ev}\langle\langle \mathit{CHANNEL} \times \mathit{VALUE} \rangle\rangle \\
\quad | & \mathit{OtherValue}
\end{array}$$

$$\begin{array}{lll}
\mathit{Exp} ::= & \mathit{Value}\langle\langle \mathit{VALUE} \rangle\rangle \\
\quad | & \mathit{Var}\langle\langle N \rangle\rangle \\
\quad | & \mathit{SeqExp}\langle\langle \mathit{SeqExpression} \rangle\rangle \\
\quad | & \mathit{FunExp}\langle\langle N \times \mathrm{seq}\, \mathit{Exp} \rangle\rangle \\
\quad | & \mathit{OtherExpression}
\end{array}$$

&

$$\begin{array}{lll}
\mathit{SeqExpression} ::= & \mathit{SeqDisplay}\langle\langle \mathrm{seq}\, \mathit{Exp} \rangle\rangle \\
\quad | & \frown\langle\langle \mathit{SeqExpression} \times \mathit{SeqExpression} \rangle\rangle \\
\quad | & \#\langle\langle \mathit{SeqExpression} \rangle\rangle \\
\quad | & \mathit{head}\langle\langle \mathit{SeqExpression} \rangle\rangle \\
\quad | & \mathit{tail}\langle\langle \mathit{SeqExpression} \rangle\rangle \\
\quad | & \mathit{last}\langle\langle \mathit{SeqExpression} \rangle\rangle \\
\quad | & \mathit{front}\langle\langle \mathit{SeqExpression} \rangle\rangle \\
\quad | & \mathit{SeqName}\langle\langle N \rangle\rangle \\
\quad | & \mathit{OtherSeqExpression}
\end{array}$$

$$EVENT == CHANNEL \times VALUE$$

$$\begin{array}{lll} Type & ::= & \mathbb{B} \\ & | & \mathbb{A} \\ & | & \mathbb{E} \\ & | & \mathbb{P}\langle\langle Type \rangle\rangle \\ & | & \mathrm{seq}\langle\langle Type \rangle\rangle \\ & | & \mathrm{dom}\langle\langle N \rangle\rangle \\ & | & \pfun \langle\langle Type \times Type \rangle\rangle \\ & | & Others \end{array}$$

$$\begin{array}{lll} CParameter & ::= & inputc\langle\langle N \rangle\rangle \\ & | & inputpc\langle\langle N \times Pred \rangle\rangle \\ & | & outputc\langle\langle Exp \rangle\rangle \\ & | & synccc\langle\langle Exp \rangle\rangle \end{array}$$

$$\begin{array}{lll} Comm & ::= & BasicComm\langle\langle CHANNEL \times \mathrm{seq}\, CParameter \rangle\rangle \\ & | & RefComm\langle\langle N \rangle\rangle \\ & | & ExpComm\langle\langle Exp \rangle\rangle \end{array}$$

$$Decl == \mathrm{seq}_1(N \times Type)$$

We assume the existence of an undefined name.

$$UNDEFINED : N$$

We also assume the existence of a function that returns the name of a *DefLHS*.

$$GetDefLHSName : DefLHS \to N$$

An schema definition can be given as a schema box or as an horizontal schema. Both have a name and a sequence of generic variables; nevertheless they differ in that the former has also a declaration part and a predicate part, and the later has also a schema expression.

$$\begin{array}{lll} ZSchemaDef & ::= & SchBox\langle\langle N \times \mathrm{seq}\, N \times Decl \times \mathrm{seq}\, Pred \rangle\rangle \\ & | & SchHor\langle\langle N \times \mathrm{seq}\, N \times SchemaExp \rangle\rangle \end{array}$$

We assume the existence of a function that returns the name of a given Z schema definition as defined below.

$$\begin{array}{|l}
GetZSchemaDefName : ZSchemaDef \rightarrow N \\
\hline
\forall name : N;\ names : \operatorname{seq} N;\ decl : Decl;\ preds : \operatorname{seq} Pred;\ sexp : SchemaExp \bullet \\
\quad GetZSchemaDefName(SchBox(name, names, decl, preds)) = name \\
\quad \land\ GetZSchemaDefName(SchHor(name, names, sexp)) = name
\end{array}$$

In a datatype definition, we have a sequence of branches; each of these branches is either a basic branch (a name) or a constructor branch (the constructor name and an expression).

$$ZBranch ::= BasicBranch\langle\!\langle N\rangle\!\rangle \quad | \quad ConstBranch\langle\!\langle N \times Exp\rangle\!\rangle$$

Finally, a Z paragraph is either a basic type (a name), or an axiomatic definition that has a declaration and a predicate part, a generic axiomatic definition that besides the declaration and predicate part has a sequence of generic variable names, or an schema, or a constant definition, or a datatype that has a sequence of branches, or finally, a predicate.

$$\begin{array}{lll}
ZParagraph ::= & & BasicType\langle\!\langle N \times \operatorname{seq} N\rangle\!\rangle \\
 & | & AxBox\langle\!\langle N \times Decl \times \operatorname{seq} Pred\rangle\!\rangle \\
 & | & GenAxBox\langle\!\langle N \times \operatorname{seq} N \times Decl \times \operatorname{seq} Pred\rangle\!\rangle \\
 & | & Schema\langle\!\langle ZSchemaDef\rangle\!\rangle \\
 & | & Const\langle\!\langle DefLHS \times Exp\rangle\!\rangle \\
 & | & Datatype\langle\!\langle N \times \operatorname{seq} ZBranch\rangle\!\rangle \\
 & | & Predicate\langle\!\langle Pred\rangle\!\rangle
\end{array}$$

We assume the existence of a function that returns the name of a given Z Paragraph as defined below.

$$\begin{array}{|l}
GetZParagraphName : ZParagraph \rightarrow N \\
\hline
\forall name : N;\ names : \operatorname{seq} N;\ decl : Decl;\ pred : Pred;\ preds : \operatorname{seq} Pred; \\
sch : ZSchemaDef;\ exp : Exp;\ branches : \operatorname{seq} ZBranch;\ defLHS : DefLHS \bullet \\
\quad GetZParagraphName(BasicType(name, names)) = name \\
\quad \land\ GetZParagraphName(AxBox(name, decl, preds)) = name \\
\quad \land\ GetZParagraphName(GenAxBox(name, names, decl, preds)) = name \\
\quad \land\ GetZParagraphName(Schema(sch)) = GetZSchemaDefName(sch) \\
\quad \land\ GetZParagraphName(Const(defLHS, exp)) = GetDefLHSName(defLHS) \\
\quad \land\ GetZParagraphName(Datatype(name, branches)) = name \\
\quad \land\ GetZParagraphName(Predicate(pred)) = UNDEFINED
\end{array}$$

This concludes the syntax of the Z part of ***Circus***. We now turn into the ***Circus***' CSP and commands part.

Before defining the syntax of an action body, we define the sets of valid arguments that are used in assignments, renaming, and alternation. The first one is composed by pairs of non-empty sequences of same length, where the first contains names and the second contains expressions; the second one is composed by pairs of non-empty sequences of same length, where both contains names.

$$\begin{aligned}
&\mathit{AssignArgs} == \{\mathit{vars} : \mathrm{seq}_1\, N;\ \mathit{exps} : \mathrm{seq}_1\, \mathit{Exp} \\
&\qquad\qquad \mid \#\mathit{vars} = \#\mathit{exps} \bullet (\mathit{vars}, \mathit{exps})\} \\
&\mathit{RenArgs} == \{\mathit{new}, \mathit{old} : \mathrm{seq}_1\, N \mid \#\mathit{new} = \#\mathit{old} \bullet (\mathit{new}, \mathit{old})\}
\end{aligned}$$

We can now define the syntax of action bodies. It corresponds to the syntactic category **Action** presented in [Oli06], but we expand the syntactic categories **SchemaExp**, **Command**, and **CSPAction**. Besides, in the definition below, we have that **ActBody** and **ParAct**, the syntactic category that corresponds to the parametrised actions, are mutually recursive; this is indicated using a & between their definitions.

$$\begin{aligned}
&\mathit{GuardedCommands} ::= \\
&\quad \mathit{GC}\langle\!\langle \{s : ((\mathrm{seq}_1\, \mathit{Pred}) \times (\mathrm{seq}_1\, \mathit{ActBody})) \mid \#s.1 = \#s.2 \bullet s\} \rangle\!\rangle \& \\
&\mathit{ActBody} ::= \mathit{ZSchExp}\langle\!\langle \mathit{SchemaExp} \rangle\!\rangle \quad \mid \quad \bullet_{\mathit{inst}\,A}\langle\!\langle \mathrm{seq}_1\, \mathit{Exp} \times \mathit{ParAct} \rangle\!\rangle \\
&\qquad \mid \ \mathit{AInst}\langle\!\langle N \rangle\!\rangle \\
&\qquad \mid \ \mathit{AInstArgs}\langle\!\langle N \times \mathrm{seq}_1\, \mathit{Exp} \rangle\!\rangle \\
&\qquad \mid \ \mathit{Skip} \ \mid \ \mathit{Stop} \ \mid \ \mathit{Chaos} \ \mid \ \rightarrow \langle\!\langle \mathit{Comm} \times \mathit{ActBody} \rangle\!\rangle \\
&\qquad \mid \ \& \ \langle\!\langle \mathit{Pred} \times \mathit{ActBody} \rangle\!\rangle \ \mid \ ;_A\langle\!\langle \mathit{ActBody} \times \mathit{ActBody} \rangle\!\rangle \\
&\qquad \mid \ \Box_A \ \langle\!\langle \mathit{ActBody} \times \mathit{ActBody} \rangle\!\rangle \ \mid \ \sqcap_A \ \langle\!\langle \mathit{ActBody} \times \mathit{ActBody} \rangle\!\rangle \\
&\qquad \mid \ \|_A \ \langle\!\langle (\mathit{NSExp} \times \mathit{CSExp} \times \mathit{NSExp}) \times \mathit{ActBody} \times \mathit{ActBody} \rangle\!\rangle \\
&\qquad \mid \ |||_A \ \langle\!\langle (\mathit{NSExp} \times \mathit{NSExp}) \times \mathit{ActBody} \times \mathit{ActBody} \rangle\!\rangle \\
&\qquad \mid \ \setminus_A \ \langle\!\langle \mathit{CSExp} \times \mathit{ActBody} \rangle\!\rangle \\
&\qquad \mid \ \mu\langle\!\langle N \times \mathit{ActBody} \rangle\!\rangle \ \mid \ \mathit{Assig}\langle\!\langle \mathit{AssignArgs} \rangle\!\rangle \\
&\qquad \mid \ \mathbf{iffi}\langle\!\langle \mathit{GuardedCommands} \rangle\!\rangle \\
&\qquad \mid \ \mathbf{var}\langle\!\langle \mathit{Decl} \times \mathit{ActBody} \rangle\!\rangle \ \mid \ \mathbf{val}\langle\!\langle \mathit{Decl} \times \mathit{ActBody} \rangle\!\rangle \\
&\qquad \mid \ \mathbf{res}\langle\!\langle \mathit{Decl} \times \mathit{ActBody} \rangle\!\rangle \ \mid \ \mathbf{vres}\langle\!\langle \mathit{Decl} \times \mathit{ActBody} \rangle\!\rangle \\
&\qquad \mid \ \mathit{SpecStmt}\langle\!\langle \mathrm{seq}\, N \times \mathit{Pred} \times \mathit{Pred} \rangle\!\rangle \\
&\qquad \mid \ \mathit{Assump}\langle\!\langle \mathit{Pred} \rangle\!\rangle \ \mid \ \mathit{Coercion}\langle\!\langle \mathit{Pred} \rangle\!\rangle \\
&\qquad \mid \ :=_A \ \langle\!\langle \mathit{RenArgs} \times \mathit{ActBody} \rangle\!\rangle \ \mid \ ;_{i\,A}\langle\!\langle \mathit{Decl} \times \mathit{ActBody} \rangle\!\rangle \\
&\qquad \mid \ \Box_{i\,A}\langle\!\langle \mathit{Decl} \times \mathit{ActBody} \rangle\!\rangle \ \mid \ \sqcap_{i\,A}\langle\!\langle \mathit{Decl} \times \mathit{ActBody} \rangle\!\rangle \\
&\qquad \mid \ \|_{i\,A}\langle\!\langle (\mathit{CSExp} \times \mathit{Decl} \times \mathit{NSExp}) \times \mathit{ActBody} \rangle\!\rangle \\
&\qquad \mid \ |||_{i\,A}\langle\!\langle (\mathit{Decl} \times \mathit{NSExp}) \times \mathit{ActBody} \rangle\!\rangle \\
&\& \\
&\mathit{ParAct} ::= \bullet_A \ \langle\!\langle \mathit{Decl} \times \mathit{ParAct} \rangle\!\rangle \ \mid \ \mathit{BAct}\langle\!\langle \mathit{ActBody} \rangle\!\rangle
\end{aligned}$$

A parametrised action is represented by the constructor $\bullet_A$; if, however, the action in not parametrised, we have a base action ($BAct$). Many of the constructors used above are subscripted with an A. This is used to differentiate between these constructors and a similar one used for processes, which are subscripted with a P. For instance, as we know, we may sequentially compose actions and processes; hence, the sequential composition of actions is represented by the constructor $;_A$.

We now present the syntax for processes. First, a process paragraph can either be a Z paragraph, an action definition that gives a name to a parametrised action, or a name set definition that gives a name to a name set expression.

$$\begin{array}{lcl} \mathit{ProcPar} & ::= & \mathit{ProcZPar}\langle\!\langle \mathit{ZParagraph} \rangle\!\rangle \\ & | & \mathit{ActDef}\langle\!\langle N \times \mathit{ParAct} \rangle\!\rangle \\ & | & \mathbf{nameset}\langle\!\langle N \times \mathit{NSExp} \rangle\!\rangle \end{array}$$

We assume the existence of a function that returns the name of a given process paragraph as defined below.

$$\begin{array}{|l} \mathit{GetProcParName} : \mathit{ProcPar} \rightarrow N \\ \hline \forall\, \mathit{zpar} : \mathit{ZParagraph};\ \mathit{name} : N;\ p : \mathit{ParAct};\ \mathit{ns} : \mathit{NSExp} \bullet \\ \quad \mathit{GetProcParName}(\mathit{ProcZPar}(\mathit{zpar})) = \mathit{GetZParagraphName}(\mathit{zpar}) \\ \quad \land\ \mathit{GetProcParName}(\mathit{ActDef}(\mathit{name}, p)) = \mathit{name} \\ \quad \land\ \mathit{GetProcParName}(\mathbf{nameset}(\mathit{name}, \mathit{ns})) = \mathit{name} \end{array}$$

Next, we have the set that contains the arguments that can be used in an explicit process definition. This set is composed by tuples $(st, ppars, main)$, where st is a Z schema definition that represents the state, $ppars$ is a sequence of process paragraph, and $main$ is an action body that represents the main action of the process.

$$\begin{array}{lcl} \mathit{ExProcDefArgs} & ::= & \mathit{Statefull}\langle\!\langle \mathit{ZSchemaDef} \times \operatorname{seq} \mathit{ProcPar} \times \mathit{ActBody} \rangle\!\rangle \\ & | & \mathit{Stateless}\langle\!\langle \operatorname{seq} \mathit{ProcPar} \times \mathit{ActBody} \rangle\!\rangle \end{array}$$

As we did for actions, we define the syntax of process bodies. It corresponds to the syntactic category Proc presented in [Oli06]. Besides, in the definition below, we have that ProcBody and ParProc, the syntactic category that corresponds to the parametrised actions, are mutually recursive; this indicated using a & between their definitions.

$$
\begin{array}{rcl}
\mathit{ProcBody} & ::= & \textbf{beginend}\ \langle\!\langle \mathit{ExProcDefArgs} \rangle\!\rangle \\
& | & ;_P \langle\!\langle \mathit{ProcBody} \times \mathit{ProcBody} \rangle\!\rangle \\
& | & \Box_P\ \langle\!\langle \mathit{ProcBody} \times \mathit{ProcBody} \rangle\!\rangle \\
& | & \sqcap_P\ \langle\!\langle \mathit{ProcBody} \times \mathit{ProcBody} \rangle\!\rangle \\
& | & \|_P\ \langle\!\langle \mathit{CSExp} \times \mathit{ProcBody} \times \mathit{ProcBody} \rangle\!\rangle \\
& | & |||_P\ \langle\!\langle \mathit{ProcBody} \times \mathit{ProcBody} \rangle\!\rangle \\
& | & \backslash_P\ \langle\!\langle \mathit{CSExp} \times \mathit{ProcBody} \rangle\!\rangle \\
& | & \bullet_{\mathit{inst}\,P} \langle\!\langle \mathrm{seq}_1\ \mathit{Exp} \times \mathit{ParProc} \rangle\!\rangle \\
& | & \odot_{\mathit{inst}} \langle\!\langle \mathrm{seq}_1\ \mathit{Exp} \times \mathit{ParProc} \rangle\!\rangle \\
& | & \mathit{PName} \langle\!\langle N \rangle\!\rangle \quad | \quad :=_P\ \langle\!\langle \mathit{RenArgs} \times \mathit{ProcBody} \rangle\!\rangle \\
& | & \textbf{ginst}\ \langle\!\langle N \times \mathrm{seq}_1\ \mathit{Exp} \rangle\!\rangle \\
& | & ;_{i\,P} \langle\!\langle \mathit{Decl} \times \mathit{ProcBody} \rangle\!\rangle \\
& | & \Box_{i\,P} \langle\!\langle \mathit{Decl} \times \mathit{ProcBody} \rangle\!\rangle \\
& | & \sqcap_{i\,P} \langle\!\langle \mathit{Decl} \times \mathit{ProcBody} \rangle\!\rangle \\
& | & \|_{i\,P} \langle\!\langle (\mathit{CSExp} \times \mathit{Decl}) \times \mathit{ProcBody} \rangle\!\rangle \\
& | & |||_{i\,P} \langle\!\langle \mathit{Decl} \times \mathit{ProcBody} \rangle\!\rangle
\end{array}
$$

&

$$
\begin{array}{rcl}
\mathit{ParProc} & ::= & \bullet_P\ \langle\!\langle \mathit{Decl} \times \mathit{ParProc} \rangle\!\rangle \\
& | & \odot \langle\!\langle \mathit{Decl} \times \mathit{ParProc} \rangle\!\rangle \\
& | & \mathit{BProc} \langle\!\langle \mathit{ProcBody} \rangle\!\rangle
\end{array}
$$

A parametrised process is represented by the constructor $\bullet_A$ and an indexing process is represented by the contructor $\odot$; if, however, the process in neither parametrised nor indexing, we have a base process (*BProc*).

Circus programs are composed by paragraphs; these can be either a Z paragraph, or a channel declaration, or a channel set declaration, or a (possibly generic) process definition, in which case, we define the process name, the sequence of generic variable names, and a (possibly parametrised) process.

$$
\begin{array}{rcl}
\mathit{ProgPar} & ::= & \mathit{ProgZPar} \langle\!\langle \mathit{ZParagraph} \rangle\!\rangle \quad | \quad \textbf{channel} \langle\!\langle \mathit{CDecl} \rangle\!\rangle \\
& | & \textbf{chanset} \langle\!\langle N \times \mathit{CSExp} \rangle\!\rangle \\
& | & \textbf{process} \langle\!\langle (N \times (\mathrm{seq}\ N)) \times \mathit{ParProc} \rangle\!\rangle
\end{array}
$$

$$\mathit{Program} == \mathrm{seq}\ \mathit{ProgPar}$$

We assume the existence of a function that returns the name of a given program paragraph as defined below.

$$
\begin{array}{|l}
GetProgParName : ProgPar \rightarrow N \\
\hline
\forall zpar : ZParagraph;\ cdecl : CDecl;\ name : N; \\
\quad names : \operatorname{seq} N;\ p : ParProc;\ cs : CSExp \bullet \\
\quad GetProgParName(ProgZPar(zpar)) = GetZParagraphName(zpar) \\
\quad \land\ GetProgParName(\mathbf{channel}(cdecl)) = UNDEFINED \\
\quad \land\ GetProgParName(\mathbf{chanset}(name, cs)) = name \\
\quad \land\ GetProgParName(\mathbf{process}((name, names), p)) = name
\end{array}
$$

H.2 Z Auxiliary Functions

function 30 leftassoc($\_ - \_$)

$$
\begin{array}{|l}
[X] \\
\hline
\_ - \_ : (\operatorname{seq} X \times \operatorname{seq} X) \rightarrow\!\!\!\!\!+ \operatorname{seq} X \\
\hline
\forall xs : \operatorname{seq} X;\ ys : \operatorname{seq} X \\
\quad \mid ys\ prefix\ xs \\
\quad \bullet\ xs\ -\ ys = squash((\operatorname{dom}\ ys) \mathbin{\lhd\!\!\!-} xs) \\
\hline
\end{array}
$$

function 30 leftassoc($\_remove\_$)

$$
\begin{array}{|l}
[X] \\
\hline
\_remove\_ : (\operatorname{seq} X \times X) \rightarrow \operatorname{seq} X \\
\hline
\forall xs : \operatorname{seq} X;\ x : X \\
\quad \bullet\ (x \in \operatorname{ran}(xs) \Rightarrow xs\ remove\ x = \\
\qquad squash(\{\min(\operatorname{dom}(xs \rhd \{x\}))\} \mathbin{\lhd\!\!\!-} xs)) \\
\quad \land\ (x \notin \operatorname{ran}(xs) \Rightarrow xs\ remove\ x = xs) \\
\hline
\end{array}
$$

function 30 leftassoc($\_ -_m \_$)

$$
\begin{array}{|l}
[X] \\
\hline
\_ -_m \_ : (\operatorname{seq} X \times \operatorname{seq} X) \rightarrow \operatorname{seq} X \\
\hline
\forall xs : \operatorname{seq} X;\ ys : \operatorname{seq} X \\
\quad \bullet\ (ys = \langle\rangle \Rightarrow xs\ -_m\ ys = xs) \\
\quad \land\ (ys \neq \langle\rangle \Rightarrow \\
\qquad xs\ -_m\ ys = (xs\ remove\ (head(ys)))\ -_m\ (tail(ys))) \\
\hline
\end{array}
$$

function 30 leftassoc$(_ \downarrow _)$

$$
\begin{array}{l}
[X] \\
\hline
\_ \downarrow \_ : (\mathrm{seq}\, X \times X) \rightarrow \mathbb{N} \\
\hline
\forall\, xs : \mathrm{seq}\, X;\ x : X \bullet xs \downarrow x = \#(xs \upharpoonright \{x\})
\end{array}
$$

function 30 leftassoc$(_ \downarrow_S _)$

$$
\begin{array}{l}
[X] \\
\hline
\_ \downarrow_S \_ : (\mathrm{seq}\, X \times \mathbb{P}\, X) \rightarrow \mathbb{N} \\
\hline
\forall\, seqs : \mathrm{seq}\, X;\ sets : \mathbb{P}\, X \bullet \\
\quad\quad \#sets = 0 \Rightarrow seqs \downarrow_S sets = 0 \\
\quad\quad \wedge\ \#sets > 0 \Rightarrow \\
\quad\quad\quad\quad \exists\, s : sets \bullet \\
\quad\quad\quad\quad\quad\quad seqs \downarrow_S sets = (seqs \downarrow s) + (seqs \downarrow_S (sets \setminus \{s\}))
\end{array}
$$

$$
\begin{array}{l}
[X] \\
\hline
replace_s : (\mathrm{seq}\, X \times (X \nrightarrow X)) \nrightarrow \mathrm{seq}\, X \\
\hline
\forall\, xs : \mathrm{seq}\, X;\ f : X \nrightarrow X \\
\quad\quad \bullet\ replace_s\,(xs, f) = \\
\quad\quad xs \oplus \{i : \mathrm{dom}(xs) \mid xs(i) \in \mathrm{dom}(f) \bullet i \mapsto f(xs(i))\}
\end{array}
$$

$$
\begin{array}{l}
[X] \\
\hline
replace_t : ((\mathbb{P}(\mathrm{seq}\, X)) \times (X \nrightarrow X)) \nrightarrow \mathbb{P}(\mathrm{seq}\, X) \\
\hline
\forall\, xs : \mathbb{P}(\mathrm{seq}\, X);\ f : X \nrightarrow X \\
\quad\quad \bullet\ replace_t\,(xs, f) = \{s : xs \bullet replace_s(s, f)\}
\end{array}
$$

$$
\begin{array}{l}
[X] \\
\hline
replace_r : ((\mathbb{P}\, X) \times (X \nrightarrow X)) \nrightarrow \mathbb{P}\, X \\
\hline
\forall\, xs : \mathbb{P}\, X;\ f : X \nrightarrow X \\
\quad\quad \bullet\ replace_r\,(xs, f) = \\
\quad\quad\quad\quad (xs \setminus \mathrm{dom}(f)) \cup \{x : xs \mid x \in \mathrm{dom}(f) \bullet f(x)\}
\end{array}
$$

$$
\begin{array}{l}
[X] \\
\hline
\mathit{replace_f} : ((\mathbb{P}(\mathrm{seq}\, X \times \mathbb{P}\, X)) \times (X \pfun X)) \pfun (\mathbb{P}(\mathrm{seq}\, X \times \mathbb{P}\, X)) \\
\hline
\forall\, \mathit{fs} : \mathbb{P}(\mathrm{seq}\, X \times \mathbb{P}\, X);\ f : X \pfun X \\
\quad \bullet\ \mathit{replace_f}\, (\mathit{fs}, f) = \\
\qquad \{\mathit{fail} : \mathit{fs} \bullet (\mathit{replace_s}(\mathit{fail}.1, f), \mathit{replace_r}(\mathit{fail}.2, f))\}
\end{array}
$$

H.3 *Circus* UTP Model

H.3.1 General Types

In Some of the definitions that follows we use the following notation:

$[\mathit{NAME}]$
$[\mathit{PREDICATE}]$
$[\mathit{TYPE}]$

$\mathit{PROCESS} == \mathit{ProcBody}$

$\checkmark : \mathit{EVENT}$

$\tau : \mathit{EVENT}$

$\mathit{BOOL\_VAL} == \{\mathit{Bool}(\mathit{TRUE}), \mathit{Bool}(\mathit{FALSE})\}$

$\mathit{INT\_VAL} == \{i : \mathbb{Z} \bullet \mathit{Int}(i)\}$

$\mathit{SEQ\_VAL} == \{s : \mathrm{seq}\ \mathit{VALUE} \bullet \mathit{Seq}(s)\}$

$\mathit{SET\_VAL} == \{s : \mathbb{P}\ \mathit{VALUE} \bullet \mathit{Set}(s)\}$

$\mathit{EVENT\_VAL} == \{e : \mathit{EVENT} \bullet \mathit{Ev}(e)\}$

$\mathit{TRACE}_{\mathcal{M}} == \{s : \mathit{SEQ\_VAL} \mid \mathrm{ran}((\mathit{Seq}^{\sim})\, s) \subseteq \mathit{EVENT\_VAL}\}$

$$REFUSAL_{\mathcal{M}} == \{s : SET\_VAL \mid ((Set^{\sim})\ s) \subseteq EVENT\_VAL\}$$

$$FAILURE_{\mathcal{M}} == \{s : TRACE_{\mathcal{M}};\ r : REFUSAL_{\mathcal{M}} \bullet Pair(s, r)\}$$

$$TRACE == \operatorname{seq} EVENT$$

$$REFUSAL == \mathbb{P}\, EVENT$$

$$FAILURE == TRACE \times REFUSAL$$

function($\{\!|\ \_\ |\!\}$)

$$\{\!|\ \_\ |\!\} : CHANNEL \to \mathbb{P}\, EVENT$$

$$\forall c : CHANNEL \bullet \{\!|\ c\ |\!\} = \{e : EVENT \mid e.1 = c\}$$

function($\{\!|\,|\ \_\ |\,|\!\}$)

$$\{\!|\,|\ \_\ |\,|\!\} : \mathbb{P}\, CHANNEL \to \mathbb{P}\, EVENT$$

$$\forall cs : \mathbb{P}\, CHANNEL \bullet \{\!|\,|\ cs\ |\,|\!\} = \bigcup\{c : cs \bullet \{\!|\ c\ |\!\}\}$$

$$production : CSExp \to \mathbb{P}\, EVENT$$

$$\begin{aligned}
&\forall cs : \mathbb{P}\, CHANNEL;\ csexp_1, csexp_2 : CSExp \bullet\\
&\quad production(CSDisplay(cs)) = \{\!|\,|\ cs\ |\,|\!\}\\
&\quad \land production(\cup(csexp_1, csexp_2)) =\\
&\qquad production(csexp_1) \cup production(csexp_2)\\
&\quad \land production(\cap(csexp_1, csexp_2)) =\\
&\qquad production(csexp_1) \cap production(csexp_2)\\
&\quad \land production(\setminus(csexp_1, csexp_2)) =\\
&\qquad production(csexp_1) \setminus production(csexp_2)
\end{aligned}$$

H.3.2 Model Auxiliary Functions

$\text{function } 30 \text{ leftassoc}(\_ -_{\mathcal{M}} \_)$

$$\_ -_{\mathcal{M}} \_ : (SEQ\_VAL \times SEQ\_VAL) \nrightarrow SEQ\_VAL$$

$$\forall xs : SEQ\_VAL;\ ys : SEQ\_VAL \\ \quad \mid ((Seq^{\sim})\ xs)\ prefix\ ((Seq^{\sim})\ ys) \\ \quad \bullet\ xs\ -_{\mathcal{M}}\ ys = Seq(((Seq^{\sim})\ xs)\ -\ ((Seq^{\sim})\ ys))$$

$\text{relation}(\_in_{\mathcal{M}}\_)$

$$\_in_{\mathcal{M}}\_ : VALUE \leftrightarrow SEQ\_VAL$$

$$\forall v : VALUE;\ s : SEQ\_VAL \\ \quad \bullet\ v\ in_{\mathcal{M}}\ s \Leftrightarrow v \in \operatorname{ran}((Seq^{\sim})\ s)$$

$\text{relation}(\_ \in_{\mathcal{M}} \_)$

$$\_ \in_{\mathcal{M}} \_ : VALUE \leftrightarrow SET\_VAL$$

$$\forall v : VALUE;\ s : SET\_VAL \\ \quad \bullet\ v\ \in_{\mathcal{M}}\ s \Leftrightarrow v \in ((Set^{\sim})\ s)$$

$\text{function } 30 \text{ leftassoc}(\_ \frown_{\mathcal{M}} \_)$

$$\_ \frown_{\mathcal{M}} \_ : (SEQ\_VAL \times SEQ\_VAL) \rightarrow SEQ\_VAL$$

$$\forall xs : SEQ\_VAL;\ ys : SEQ\_VAL \\ \quad \bullet\ xs\ \frown_{\mathcal{M}}\ ys = Seq(((Seq^{\sim})\ xs)\ \frown\ ((Seq^{\sim})\ ys))$$

$$\mathcal{C}_{trace} : TRACE_{\mathcal{M}} \rightarrow TRACE$$

$$\forall t : TRACE_{\mathcal{M}} \\ \quad \bullet\ \mathcal{C}_{trace}\ t = \{i : \operatorname{dom}((Seq^{\sim})\ t) \bullet i \mapsto ((Ev^{\sim})(((Seq^{\sim})\ t)(i)))\}$$

$$\mathcal{C}_{traces} : \mathbb{P}\ TRACE_{\mathcal{M}} \rightarrow \mathbb{P}\ TRACE$$

$$\forall ts : \mathbb{P}\ TRACE_{\mathcal{M}} \\ \quad \bullet\ \mathcal{C}_{traces}\ ts = \{t : ts \bullet \mathcal{C}_{trace}\ t\}$$

$$\mathcal{C}_{\mathit{refusal}} : \mathit{REFUSAL}_{\mathcal{M}} \rightarrow \mathit{REFUSAL}$$
$$\forall r : \mathit{REFUSAL}_{\mathcal{M}} \\ \quad \bullet \mathcal{C}_{\mathit{refusal}}\ r = \{i : ((\mathit{Set}^{\sim})\ r) \bullet ((\mathit{Ev}^{\sim}) r)\}$$

$$\mathcal{C}_{\mathit{failure}} : \mathit{FAILURE}_{\mathcal{M}} \rightarrow \mathit{FAILURE}$$
$$\forall f : \mathit{FAILURE}_{\mathcal{M}} \\ \quad \bullet \mathcal{C}_{\mathit{failure}}\ f = (\mathcal{C}_{\mathit{trace}}\ (((\mathit{Pair}^{\sim})\ f).1), \mathcal{C}_{\mathit{refusal}}\ (((\mathit{Pair}^{\sim})\ f).2))$$

$$\mathcal{C}_{\mathit{failures}} : \mathbb{P}\ \mathit{FAILURE}_{\mathcal{M}} \rightarrow \mathbb{P}\ \mathit{FAILURE}$$
$$\forall \mathit{fs} : \mathbb{P}\ \mathit{FAILURE}_{\mathcal{M}} \\ \quad \bullet \mathcal{C}_{\mathit{failures}}\ \mathit{fs} = \{f : \mathit{fs} \bullet \mathcal{C}_{\mathit{failure}}\ f\}$$

function 30 leftassoc($\_ \cup_{\mathcal{M}} \_$)

$$\_ \cup_{\mathcal{M}} \_ : (\mathit{SET\_VAL} \times \mathit{SET\_VAL}) \rightarrow \mathit{SET\_VAL}$$
$$\forall \mathit{xs} : \mathit{SET\_VAL};\ \mathit{ys} : \mathit{SET\_VAL} \\ \quad \bullet \mathit{xs} \cup_{\mathcal{M}} \mathit{ys} = \mathit{Set}(((\mathit{Set}^{\sim})\ \mathit{xs}) \cup ((\mathit{Set}^{\sim})\ \mathit{ys}))$$

H.3.3 Predicate Model

$$\mathit{Model} == \mathit{NAME} \twoheadrightarrow\!\!\!\!\!\!\shortmid\ \ \mathit{VALUE}$$

$$\mathit{true}_{\mathcal{M}} : \mathbb{P}\ \mathit{Model}$$
$$\mathit{true}_{\mathcal{M}} = \mathit{Model}$$

$$\neg_{\mathcal{M}} : \mathbb{P}\ \mathit{Model} \rightarrow \mathbb{P}\ \mathit{Model}$$
$$\forall m_1 : \mathbb{P}\ \mathit{Model} \bullet \neg_{\mathcal{M}} m_1 = \mathit{Model} \setminus m_1$$

function 40 leftassoc($\_ \wedge_{\mathcal{M}} \_$)

$$\_ \wedge_{\mathcal{M}} \_ : \mathbb{P}\ \mathit{Model} \times \mathbb{P}\ \mathit{Model} \rightarrow \mathbb{P}\ \mathit{Model}$$
$$\forall m_1, m_2 : \mathbb{P}\ \mathit{Model} \bullet m_1 \wedge_{\mathcal{M}} m_2 = m_1 \cap m_2$$

function 40 leftassoc($\_ \vee_{\mathcal{M}} \_$)

$$\_ \vee_{\mathcal{M}} \_ : \mathbb{P}\ \mathit{Model} \times \mathbb{P}\ \mathit{Model} \rightarrow \mathbb{P}\ \mathit{Model}$$
$$\forall m_1, m_2 : \mathbb{P}\ \mathit{Model} \bullet m_1 \vee_{\mathcal{M}} m_2 = m_1 \cup m_2$$

H.3.4 Observational Variables

$$\begin{array}{|l}
ok, ok' : NAME \\
OK, OK' : \mathbb{P}\, Model \\
\hline
\forall\, m : Model \bullet \{m(ok), m(ok')\} \subseteq BOOL\_VAL \\
OK = \{m : Model \mid m(ok) = Bool(TRUE)\} \\
OK' = \{m : Model \mid m(ok') = Bool(TRUE)\}
\end{array}$$

$$\begin{array}{|l}
wt, wt' : NAME \\
WT, WT' : \mathbb{P}\, Model \\
\hline
\forall\, m : Model \bullet \{m(wt), m(wt')\} \subseteq BOOL\_VAL \\
WT = \{m : Model \mid m(wt) = Bool(TRUE)\} \\
WT' = \{m : Model \mid m(wt') = Bool(TRUE)\}
\end{array}$$

$$\begin{array}{|l}
tr, tr' : NAME \\
\hline
\forall\, m : Model \bullet \\
\quad \{m(tr), m(tr')\} \subseteq TRACE_{\mathcal{M}} \\
\quad \wedge\ ((Seq^{\sim})\,(m(tr)))\ prefix\ ((Seq^{\sim})\,(m(tr')))
\end{array}$$

$$\begin{array}{|l}
ref, ref' : NAME \\
\hline
\forall\, m : Model \bullet \{m(ref), m(ref')\} \subseteq REFUSAL_{\mathcal{M}}
\end{array}$$

H.3.5 Semantic Functions

function($[\![\_]\!]^{\mathcal{C}}$)

$$\begin{array}{|l}
[\![\_]\!]^{\mathcal{C}} : PROCESS \rightarrow PREDICATE
\end{array}$$

function($[\![\_]\!]^{\mathcal{P}}$)

$$\begin{array}{|l}
[\![\_]\!]^{\mathcal{P}} : PREDICATE \rightarrow \mathbb{P}\, Model
\end{array}$$

$$\begin{array}{|l}
Circus\_Healthy : \mathbb{P}\, PROCESS \\
\hline
\forall\, p : PROCESS \bullet \\
\quad p \in Circus\_Healthy \\
\quad \Leftrightarrow \\
\quad \forall\, m : [\![([\![p]\!]^{\mathcal{C}})]\!]^{\mathcal{P}} \bullet \\
\quad\quad ((Seq^{\sim})\,(m(tr)))\ prefix\ ((Seq^{\sim})\,(m(tr')))
\end{array}$$

$$\begin{array}{|l} CIRCUS\_PROCESS : \mathbb{P}\, PROCESS \\ \hline \forall\, p : PROCESS \bullet \\ \quad p \in CIRCUS\_PROCESS \\ \quad \Leftrightarrow \\ \quad p \in Circus\_Healthy \end{array}$$

H.4 Linking UTP Model to FD Model

$$\begin{array}{|l} Normal : CIRCUS\_PROCESS \rightarrow \mathbb{P}\, Model \\ \hline \forall\, P : CIRCUS\_PROCESS \bullet \\ \quad Normal(P) = OK \wedge_{\mathcal{M}} \neg_{\mathcal{M}} WT \wedge_{\mathcal{M}} OK' \wedge_{\mathcal{M}} [\![(\![[\![P]\!]^{\mathcal{C}}]\!)]\!]^{\mathcal{P}} \end{array}$$

$$\begin{array}{|l} Terminate : CIRCUS\_PROCESS \rightarrow \mathbb{P}\, Model \\ \hline \forall\, P : CIRCUS\_PROCESS \bullet Terminate(P) = Normal(P) \wedge_{\mathcal{M}} \neg_{\mathcal{M}} WT' \end{array}$$

$$\begin{array}{|l} Diverge : CIRCUS\_PROCESS \rightarrow \mathbb{P}\, Model \\ \hline \forall\, P : CIRCUS\_PROCESS \bullet Diverge(P) = \\ \quad OK \wedge_{\mathcal{M}} \neg_{\mathcal{M}} WT \wedge_{\mathcal{M}} \neg_{\mathcal{M}} OK' \wedge_{\mathcal{M}} [\![(\![[\![P]\!]^{\mathcal{C}}]\!)]\!]^{\mathcal{P}} \end{array}$$

$$\begin{array}{|l} traces_{\mathcal{M}} : CIRCUS\_PROCESS \rightarrow \mathbb{P}\, TRACE_{\mathcal{M}} \\ \hline \forall\, P : CIRCUS\_PROCESS \bullet \\ \quad traces_{\mathcal{M}}(P) = \\ \qquad \{m : Normal(P) \bullet m(tr') -_{\mathcal{M}} m(tr)\} \\ \qquad \cup \{m : Normal(P) \bullet \\ \qquad\quad (m(tr') -_{\mathcal{M}} m(tr)) \frown_{\mathcal{M}} Seq(\langle Ev(\checkmark)\rangle)\} \end{array}$$

$$\begin{array}{|l} traces : CIRCUS\_PROCESS \rightarrow \mathbb{P}\, TRACE \\ \hline \forall\, P : CIRCUS\_PROCESS \bullet traces(P) = \mathcal{C}_{traces}\,(traces_{\mathcal{M}}(P)) \end{array}$$

$$\begin{array}{|l} divergences_{\mathcal{M}} : CIRCUS\_PROCESS \rightarrow \mathbb{P}\, TRACE_{\mathcal{M}} \\ \hline \forall\, P : CIRCUS\_PROCESS \bullet \\ \quad divergences_{\mathcal{M}}(P) = \{m : Diverge(P) \bullet m(tr') -_{\mathcal{M}} m(tr)\} \end{array}$$

$$\begin{array}{|l} divergences : CIRCUS\_PROCESS \rightarrow \mathbb{P}\, TRACE \\ \hline \forall P : CIRCUS\_PROCESS \bullet divergences(P) = \mathcal{C}_{traces}\, (divergences_{\mathcal{M}}(P)) \end{array}$$

$$\begin{array}{|l} traces_{\perp} : CIRCUS\_PROCESS \rightarrow \mathbb{P}\, TRACE \\ \hline \forall P : CIRCUS\_PROCESS \bullet traces_{\perp}(P) = traces(P) \cup divergences(P) \end{array}$$

$$\begin{array}{|l} failures_{\mathcal{M}} : CIRCUS\_PROCESS \rightarrow \mathbb{P}\, FAILURE_{\mathcal{M}} \\ \hline \forall P : CIRCUS\_PROCESS \bullet \\ \quad failures_{\mathcal{M}}(P) = \\ \qquad \{m : Normal(P) \bullet Pair(m(tr') -_{\mathcal{M}} m(tr), m(ref'))\} \\ \qquad \cup \{m : Normal(P) \wedge_{\mathcal{M}} WT' \bullet \\ \qquad\quad Pair(m(tr') -_{\mathcal{M}} m(tr), m(ref') \cup_{\mathcal{M}} Set(\{Ev(\checkmark)\}))\} \\ \qquad \cup \{m : Terminate(P) \bullet \\ \qquad\quad Pair(m(tr') \frown_{\mathcal{M}} Seq(\langle Ev(\checkmark)\rangle), m(ref'))\} \\ \qquad \cup \{m : Terminate(P) \bullet \\ \qquad\quad Pair(m(tr') \frown_{\mathcal{M}} Seq(\langle Ev(\checkmark)\rangle), m(ref') \cup_{\mathcal{M}} Set(\{Ev(\checkmark)\}))\} \end{array}$$

$$\begin{array}{|l} failures : CIRCUS\_PROCESS \rightarrow \mathbb{P}\, FAILURE \\ \hline \forall P : CIRCUS\_PROCESS \bullet failures(P) = \mathcal{C}_{failures}\, (failures_{\mathcal{M}}(P)) \end{array}$$

The following definition was not in [CG10], but it is based on a similar definition from [CW06].

$$failures_{\perp}(A) = failures(A) \cup \{(s, ref) \mid s \in divergences(P) \underline{\wedge\ ref \in \Sigma^{*\checkmark}}\}$$

$$\begin{array}{|l} failures_{\perp} : CIRCUS\_PROCESS \rightarrow \mathbb{P}\, FAILURE \\ \hline \forall P : CIRCUS\_PROCESS \bullet failures_{\perp}(P) = \\ \quad failures(P) \cup \{s : divergences(P);\ r : \mathbb{P}\, EVENT \bullet (s, r)\} \end{array}$$

H.5 Properties

$$\begin{array}{|l} DeadlockFree : \mathbb{P}\, CIRCUS\_PROCESS \\ \hline \forall p : CIRCUS\_PROCESS \bullet \\ \quad p \in DeadlockFree \\ \quad \Leftrightarrow \\ \quad \forall s : traces(p) \bullet (s, EVENT) \notin failures(p) \end{array}$$

$$
\begin{array}{|l}
\mathit{InfTraces} : \mathbb{P}\ \mathit{CIRCUS\_PROCESS} \\
\hline
\forall p : \mathit{CIRCUS\_PROCESS} \bullet \\
\qquad p \in \mathit{InfTraces} \Leftrightarrow (\mathit{traces}(p) \notin \mathbb{F}\ \mathit{TRACE})
\end{array}
$$

$$
\begin{array}{|l}
\mathit{DivergenceFree} : \mathbb{P}\ \mathit{CIRCUS\_PROCESS} \\
\hline
\forall p : \mathit{CIRCUS\_PROCESS} \bullet \\
\qquad p \in \mathit{DivergenceFree} \Leftrightarrow (\mathit{divergences}(p) = \emptyset)
\end{array}
$$

$$
\begin{array}{|l}
\mathit{Deterministic} : \mathbb{P}\ \mathit{CIRCUS\_PROCESS} \\
\hline
\forall p : \mathit{CIRCUS\_PROCESS} \bullet \\
\qquad p \in \mathit{Deterministic} \\
\qquad \Leftrightarrow \\
\qquad (p \in \mathit{DivergenceFree} \\
\qquad\qquad \land \forall t : \mathit{TRACE};\ a : \mathit{EVENT} \bullet \\
\qquad\qquad\qquad t \frown \langle a \rangle \in \mathit{traces}(p) \Rightarrow \\
\qquad\qquad\qquad\qquad (t, \{a\}) \notin \mathit{failures}(p))
\end{array}
$$

H.6 Refinement

relation($\_ \sqsubseteq_{\mathrm{T}} \_$)

$$
\begin{array}{|l}
\_ \sqsubseteq_{\mathrm{T}} \_ : \mathit{CIRCUS\_PROCESS} \leftrightarrow \mathit{CIRCUS\_PROCESS} \\
\hline
\forall p_1, p_2 : \mathit{CIRCUS\_PROCESS} \bullet \\
\qquad p_1 \sqsubseteq_{\mathrm{T}} p_2 \\
\qquad \Leftrightarrow \\
\qquad \mathit{traces}(p_2) \subseteq \mathit{traces}(p_1)
\end{array}
$$

relation($\_ \equiv_{\mathrm{T}} \_$)

$$
\begin{array}{|l}
\_ \equiv_{\mathrm{T}} \_ : \mathit{CIRCUS\_PROCESS} \leftrightarrow \mathit{CIRCUS\_PROCESS} \\
\hline
\forall p_1, p_2 : \mathit{CIRCUS\_PROCESS} \bullet \\
\qquad p_1 \equiv_{\mathrm{T}} p_2 \\
\qquad \Leftrightarrow \\
\qquad (p_1 \sqsubseteq_{\mathrm{T}} p_2 \land p_2 \sqsubseteq_{\mathrm{T}} p_1)
\end{array}
$$

relation($\_ \sqsubseteq_{\mathrm{F}} \_$)

$$\begin{array}{|l}
\_ \sqsubseteq_{\mathrm{F}} \_ : \mathit{CIRCUS\_PROCESS} \leftrightarrow \mathit{CIRCUS\_PROCESS} \\
\hline
\forall p_1, p_2 : \mathit{CIRCUS\_PROCESS} \bullet \\
\qquad p_1 \sqsubseteq_{\mathrm{F}} p_2 \\
\qquad \Leftrightarrow \\
\qquad \mathit{failures}(p_2) \subseteq \mathit{failures}(p_1)
\end{array}$$

relation($\_ \equiv_{\mathrm{F}} \_$)

$$\begin{array}{|l}
\_ \equiv_{\mathrm{F}} \_ : \mathit{CIRCUS\_PROCESS} \leftrightarrow \mathit{CIRCUS\_PROCESS} \\
\hline
\forall p_1, p_2 : \mathit{CIRCUS\_PROCESS} \bullet \\
\qquad p_1 \equiv_{\mathrm{F}} p_2 \\
\qquad \Leftrightarrow \\
\qquad (p_1 \sqsubseteq_{\mathrm{F}} p_2 \land p_2 \sqsubseteq_{\mathrm{F}} p_1)
\end{array}$$

relation($\_ \sqsubseteq_{\mathrm{FD}} \_$)

$$\begin{array}{|l}
\_ \sqsubseteq_{\mathrm{FD}} \_ : \mathit{CIRCUS\_PROCESS} \leftrightarrow \mathit{CIRCUS\_PROCESS} \\
\hline
\forall p_1, p_2 : \mathit{CIRCUS\_PROCESS} \bullet \\
\qquad p_1 \sqsubseteq_{\mathrm{FD}} p_2 \\
\qquad \Leftrightarrow \\
\qquad (\mathit{failures}_{\perp}(p_2) \subseteq \mathit{failures}_{\perp}(p_1) \land \mathit{divergences}(p_2) \subseteq \mathit{divergences}(p_2))
\end{array}$$

relation($\_ \equiv_{\mathrm{FD}} \_$)

$$\begin{array}{|l}
\_ \equiv_{\mathrm{FD}} \_ : \mathit{CIRCUS\_PROCESS} \leftrightarrow \mathit{CIRCUS\_PROCESS} \\
\hline
\forall p_1, p_2 : \mathit{CIRCUS\_PROCESS} \bullet \\
\qquad p_1 \equiv_{\mathrm{FD}} p_2 \\
\Leftrightarrow \\
(p_1 \sqsubseteq_{\mathrm{FD}} p_2 \land p_2 \sqsubseteq_{\mathrm{FD}} p_1)
\end{array}$$

relation($\_ \sqsubseteq_{\mathrm{P}} \_$)

$$
\begin{array}{|l}
\_ \sqsubseteq_{\mathrm{P}} \_ : \mathit{CIRCUS\_PROCESS} \leftrightarrow \mathit{CIRCUS\_PROCESS} \\
\hline
\forall p_1, p_2 : \mathit{CIRCUS\_PROCESS} \\
\quad \mid \mathit{divergences}(p_1) = \mathit{divergences}(p_2) = \emptyset \\
\quad \bullet\ p_1 \sqsubseteq_{\mathrm{P}} p_2 \\
\quad \Leftrightarrow \\
\quad (p_1 \sqsubseteq_{\mathrm{F}} p_2 \land p_1 \sqsubseteq_{\mathrm{FD}} p_2)
\end{array}
$$

relation($\_ \equiv_{\mathrm{P}} \_$)

$$
\begin{array}{|l}
\_ \equiv_{\mathrm{P}} \_ : \mathit{CIRCUS\_PROCESS} \leftrightarrow \mathit{CIRCUS\_PROCESS} \\
\hline
\forall p_1, p_2 : \mathit{CIRCUS\_PROCESS} \bullet \\
\quad p_1 \equiv_{\mathrm{P}} p_2 \\
\Leftrightarrow \\
(p_1 \sqsubseteq_{\mathrm{P}} p_2 \land p_2 \sqsubseteq_{\mathrm{P}} p_1)
\end{array}
$$

I Z Formalisation of *Circus* BRIC

section *circus_bric_toolkit* parents *csp_circus_toolkit*

I.1 Basic Definitions

The function α returns the alphabet of events of a given process.

$$
\begin{array}{|l}
\alpha : \mathit{CIRCUS\_PROCESS} \rightarrow \mathbb{P}\,\mathit{EVENT} \\
\hline
\forall p : \mathit{CIRCUS\_PROCESS} \bullet \alpha\, p = \bigcup\{t : \mathit{traces}(p) \bullet \mathrm{ran}(t)\}
\end{array}
$$

$$
\begin{array}{|l}
\mathit{inputs}, \mathit{outputs} : \mathit{CHANNEL} \times \mathit{CIRCUS\_PROCESS} \rightarrow \mathbb{P}\,\mathit{EVENT} \\
\hline
\forall c : \mathit{CHANNEL};\ p : \mathit{CIRCUS\_PROCESS} \bullet \\
\quad \mathit{inputs}(c, p) \subseteq \{\!|\ c\ |\!\} \\
\quad \land\ \mathit{outputs}(c, p) \subseteq \{\!|\ c\ |\!\}
\end{array}
$$

$$
\begin{array}{|l}
\mathit{inputs}_P, \mathit{outputs}_P : \mathit{CIRCUS\_PROCESS} \rightarrow \mathbb{P}\,\mathit{EVENT} \\
\hline
\forall p : \mathit{CIRCUS\_PROCESS} \bullet \\
\quad \mathit{inputs}_P(p) = \bigcup\{c : \mathit{CHANNEL} \bullet \mathit{inputs}(c, p)\} \\
\quad \land\ \mathit{outputs}_P(p) = \bigcup\{c : \mathit{CHANNEL} \bullet \mathit{outputs}(c, p)\}
\end{array}
$$

$$
\begin{array}{|l}
\mathit{channel} : \mathit{EVENT} \rightarrow \mathit{CHANNEL} \\
\hline
\forall e : \mathit{EVENT} \bullet \mathit{channel}(e) = e.1
\end{array}
$$

$$
\begin{array}{|l}
\mathit{channels} : \mathbb{P}\,\mathit{EVENT} \rightarrow \mathbb{P}\,\mathit{CHANNEL} \\
\hline
\forall es : \mathbb{P}\,\mathit{EVENT} \bullet \mathit{channels}(es) = \{e : es \bullet \mathit{channel}(e)\}
\end{array}
$$

function 30 leftassoc(_/_)

$$
\begin{array}{|l}
\_/\_ : (\mathit{CIRCUS\_PROCESS} \times \mathit{TRACE}) \pfun \mathit{CIRCUS\_PROCESS} \\
\hline
\forall p : \mathit{CIRCUS\_PROCESS};\ s : \mathit{TRACE} \\
\quad\quad \mid s \in \mathit{traces}(p) \\
\quad\quad \bullet\ \mathit{traces}(p/s) = \{t : \mathit{TRACE} \mid s \frown t \in \mathit{traces}(p)\} \\
\quad\quad \land\ \mathit{failures}(p/s) = \\
\quad\quad\quad\quad \{f : \mathit{failures}(p);\ t : \mathit{TRACE} \mid f.1 = s \frown t \bullet (t, f.2)\} \\
\quad\quad \land\ \mathit{divergences}(p/s) = \{t : \mathit{TRACE} \mid s \frown t \in \mathit{divergences}(p)\}
\end{array}
$$

I.2 Component Model

I.2.1 I/O channels

Definition I.1 (I/O channels) *We say a channel c is an I/O channel if there exists two functions, $inputs(c, P)$ and $outputs(c, P)$, for every process P, such that:*

$$
\begin{array}{l}
\mathit{inputs}(c, P) \cup \mathit{outputs}(c, P) \subseteq \{\!|\ c\ |\!\} \\
\land\ \mathit{inputs}(c, P) \cap \mathit{outputs}(c, P) = \emptyset
\end{array}
$$

Formally

$$
\begin{array}{|l}
\mathit{IOChannels} : \mathbb{P}\,\mathit{CHANNEL} \\
\hline
\forall c : \mathit{CHANNEL} \bullet \\
\quad\quad c \in \mathit{IOChannels} \\
\quad\quad \Leftrightarrow \\
\quad\quad (\forall p : \mathit{CIRCUS\_PROCESS} \bullet \\
\quad\quad\quad\quad \mathit{inputs}(c, p) \cup \mathit{outputs}(c, p) \subseteq \{\!|\ c\ |\!\} \\
\quad\quad\quad\quad \land\ \mathit{inputs}(c, p) \cap \mathit{outputs}(c, p) = \emptyset)
\end{array}
$$

I.2.2 Input determinism

Definition 3.11 (Input determinism) *We say a process P is input deterministic if*

$$\forall\, s \frown \langle c.a\rangle : traces(P) \mid\ c.a \in inputs(c, P) \bullet (s, \{c.a\}) \notin failures(P)$$

Formally

$$\begin{array}{|l}
inputdet : \mathbb{P}\ CIRCUS\_PROCESS \\
\hline
\forall\, p : CIRCUS\_PROCESS \bullet \\
\quad p \in inputdet \\
\quad \Leftrightarrow \\
\quad (\forall\, e : EVENT;\ s : TRACE \\
\qquad \mid s \frown \langle e\rangle \in traces(p) \land e \in inputs_P(p) \\
\qquad \bullet (s, \{e\}) \notin failures(p))
\end{array}$$

I.2.3 Strong output decisiveness

Definition 3.12 (Strong output decisiveness) *We say a process P is strong output decisive if:*

$$\begin{array}{l}
\forall\, s \frown \langle c.b\rangle : traces(P) \mid c.b \in outputs(c, P) \bullet \\
\quad (s, outputs(c, P)) \notin failures(P) \\
\quad \land (s, outputs(c, P) \setminus \{c.b\}) \in failures(P)
\end{array}$$

Formally

$$\begin{array}{|l}
strongoutputdec : \mathbb{P}\ CIRCUS\_PROCESS \\
\hline
\forall\, p : CIRCUS\_PROCESS \bullet \\
\quad p \in strongoutputdec \\
\quad \Leftrightarrow \\
\quad (\forall\, e : EVENT;\ s : TRACE \\
\qquad \mid s \frown \langle e\rangle \in traces(p) \land e \in outputs_P(p) \\
\qquad \bullet (\exists\, c : CHANNEL \\
\qquad\quad \mid e \in outputs(c, p) \\
\qquad\quad \bullet (s, outputs(c, p)) \notin failures(p) \\
\qquad\qquad \land (s, outputs(c, p) \setminus \{e\}) \in failures(p)))
\end{array}$$

I.2.4 I/O Process

New Definition 4.2 (I/O process) *We say P is an I/O process if:*

- *P only uses I/O Channels and $\alpha P = inputs(P) \cup outputs(P)$;*
- *P has infinite traces;*
- *P is divergent-free;*
- *P is input deterministic;*
- *P is strong output decisive.*

Formally

$$
\begin{array}{|l}
IOProcess : \mathbb{P}\ CIRCUS\_PROCESS \\
\hline
\forall p : CIRCUS\_PROCESS \bullet \\
\quad p \in IOProcess \\
\quad \Leftrightarrow \\
\quad (\{e : \alpha(p) \bullet channel(e)\} \subseteq IOChannels \\
\qquad \land\ \alpha(p) = inputs_P(p) \cup outputs_P(p) \\
\qquad \land\ p \in InfTraces \\
\qquad \land\ p \in DivergenceFree \\
\qquad \land\ p \in inputdet \\
\qquad \land\ p \in strongoutputdec)
\end{array}
$$

I.2.5 Component Contract

Definition 4.1 (Component contract) *A component contract Ctr comprises an observational behaviour $\mathcal{B}$ (a* **Circus** *process), a set of communication channels $\mathcal{C}$, a set of interfaces $\mathcal{I}$, and a total function $\mathcal{R} : \mathcal{C} \rightarrow \mathcal{I}$ between channels and interfaces:*

$$Ctr : \langle \mathcal{B}, \mathcal{R}, \mathcal{I}, \mathcal{C} \rangle$$

such that

- *$\mathcal{B}$ is an I/O process*
- $\operatorname{dom} \mathcal{R} = \mathcal{C} \wedge \operatorname{ran} \mathcal{R} = \mathcal{I}$

Formally

$$\begin{array}{|l}
CTR : \mathbb{P}(CIRCUS\_PROCESS \times (CHANNEL \rightarrow TYPE) \times \\
\qquad\qquad (\mathbb{P}\, TYPE) \times (\mathbb{P}\, CHANNEL)) \\
\hline
\forall B : CIRCUS\_PROCESS;\ R : CHANNEL \rightarrow TYPE; \\
\quad I : \mathbb{P}\, TYPE;\ C : \mathbb{P}\, CHANNEL \bullet \\
\qquad (B, R, I, C) \in CTR \\
\qquad \Leftrightarrow \\
\qquad B \in IOProcess \wedge \operatorname{dom} R = C \wedge \operatorname{ran} R = I
\end{array}$$

$$\begin{array}{|l}
\mathcal{B} : CTR \rightarrow CIRCUS\_PROCESS \\
\hline
\forall p : CTR \bullet \mathcal{B}(p) = p.1
\end{array}$$

$$\begin{array}{|l}
\mathcal{R} : CTR \rightarrow (CHANNEL \rightarrow TYPE) \\
\hline
\forall p : CTR \bullet \mathcal{R}(p) = p.2
\end{array}$$

$$\begin{array}{|l}
\mathcal{I} : CTR \rightarrow \mathbb{P}\, TYPE \\
\hline
\forall p : CTR \bullet \mathcal{I}(p) = p.3
\end{array}$$

$$\begin{array}{|l}
\mathcal{C} : CTR \rightarrow \mathbb{P}\, CHANNEL \\
\hline
\forall p : CTR \bullet \mathcal{C}(p) = p.4
\end{array}$$

I.2.6 Asynchronous Composition

Definition I.2 (Asynchronous Composition)

$$P_S = |||_{P \in S} P$$

$$AsyncComp(S, F) = P_S \parallel_{\mathrm{dom}\ F} \left(|||_{c \in \mathrm{dom}\ F} BUFF^{\infty}_{IO}(c, F(c)) \right)$$

$$\begin{aligned} BUFF^{1}_{IO}(c, z) &= BUFF_{IO}(COPY, R^{c \to z}_{IO}, R^{z \to c}_{IO}) \\ BUFF^{n}_{IO}(c, z) &= BUFF_{IO}(B^{n}, R^{c \to z}_{IO}, R^{z \to c}_{IO}) \\ BUFF^{\infty}_{IO}(c, z) &= BUFF_{IO}(B^{\infty}_{\langle\rangle}, R^{c \to z}_{IO}, R^{z \to c}_{IO}) \end{aligned}$$

$$BUFF_{IO}(BF, LR1, LR2) = (BF(LR1) \ ||| \ BF(LR2))$$

$$COPY(LR) = ?x : \mathrm{dom}\, LR \to LR(x) \to COPY(LR)$$

$$LR = \{ left.x \mapsto right.x \mid x \in T \}$$

$$\begin{aligned} &B^{\infty}_{\langle\rangle}(LR) = ?x : \mathrm{dom}\, LR \to B^{\infty}_{\langle x \rangle} \\ &B^{\infty}_{s \frown \langle y \rangle}(LR) = (?x : \mathrm{dom}\, LR \to B^{\infty}_{\langle x \rangle \frown s \frown \langle y \rangle}(LR)) \ \Box \ (LR(y) \to B^{\infty}_{s}(LR)) \end{aligned}$$

$$B^{n}(LR) = COPY(LR) \ \rangle\!\rangle_{LR} COPY(LR) \ \rangle\!\rangle_{LR} \ldots \ \rangle\!\rangle_{LR} COPY(LR)$$

$$P \ \rangle\!\rangle_{LR} Q = (P \, [\![RM]\!] \parallel_{mid} Q \, [\![LM]\!]) \setminus mid$$

where LR, RM and LM are bijections and mid is a set of events, such that:

- $(\alpha P \cup \alpha Q) \subseteq (\mathrm{dom}\, LR \cup \mathrm{ran}\, LR)$
- $(\mathrm{dom}\, LR \cap \mathrm{ran}\, LR) = \emptyset$
- $(\mathrm{dom}\, LR \cup \mathrm{ran}\, LR) \cap mid = \emptyset$
- $\mathrm{ran}\, RM = mid$
- $\mathrm{ran}\, ML = mid$
- $\mathrm{dom}\, RM = \mathrm{ran}\, LR$
- $\mathrm{dom}\, LM = \mathrm{dom}\, LR.$

Formally For simplicity we slightly changed $AsyncComp$.

- It receive as arguments two processes instead of a set of process
- The mapping is no longer from channel to channel, but from pairs to pairs (a pair has a process and a channel)

This makes it easier to correctly use the renaming function $P \llbracket R_{IO}^{a \to b} \rrbracket$ used in the definition, sice its definition is now parameterised by the process to which it is being applied.

$$
\begin{array}{|l}
BR_{IO} : (CIRCUS\_PROCESS \times CHANNEL) \times (CIRCUS\_PROCESS \times CHANNEL) \\
\qquad \to (EVENT \to EVENT) \\
\hline
\forall P, Q : CIRCUS\_PROCESS;\ c, z : CHANNEL \bullet \\
\quad BR_{IO}((P, c), (Q, z)) = \\
\qquad \{e : outputs_P(P) \mid channel(e) = c \bullet e \mapsto (z, e.2)\}
\end{array}
$$

We also need to change and formalise the definition of $BUFF_{IO}^{\infty}$. The *Circus* processes are:

$$
\begin{array}{l}
\textbf{process}\ BUFF_{IO}^{\infty} \mathrel{\widehat=} \\
\quad pc, qz : CIRCUS\_PROCESS \times CHANNEL \bullet \\
\qquad B^{\infty}(\langle\rangle, BR_{IO}(pc, pz)) \interleave B^{\infty}(\langle\rangle, BR_{IO}(pz, pc)) \\
\textbf{process}\ B^{\infty} \mathrel{\widehat=} \\
\quad s : TRACE;\ LR : EVENT \to EVENT \bullet \\
\quad \textbf{begin} \\
\qquad \bullet\ (\Box\, x : \mathrm{dom}\, LR \bullet x \to B^{\infty}(\langle x\rangle \frown s, LR)) \\
\qquad\ \ \Box\ (\#s > 0)\ \&\ LR(last(s)) \to B^{\infty}(front(s), LR) \\
\quad \textbf{end} \\
\textbf{process}\ B^{n} \mathrel{\widehat=} \\
\quad n : \mathbb{N};\ s : TRACE;\ LR : EVENT \to EVENT \bullet \\
\quad \textbf{begin} \\
\qquad \bullet\ (\#s < n)\ \&\ (\Box\, x : \mathrm{dom}\, LR \bullet x \to B^{\infty}(n+1, \langle x\rangle \frown s, LR)) \\
\qquad\ \ \Box\ (\#s > 0)\ \&\ LR(last(s)) \to B^{\infty}(n-1, front(s), LR) \\
\quad \textbf{end}
\end{array}
$$

$$
\begin{array}{l}
\textbf{process}\ AsyncComp \mathrel{\widehat=} \\
\quad P, Q : CIRCUS\_PROCESS; \\
\quad F : (CIRCUS\_PROCESS \times CHANNEL) \to (CIRCUS\_PROCESS \times CHANNEL) \bullet \\
\qquad (P \interleave Q) \\
\qquad \llbracket\{\!|\ \underline{\alpha\, P \cup \alpha\, Q}\ |\!\}\rrbracket \\
\qquad (\interleave\ \overline{pc \in \mathrm{dom}\ F} \bullet BUFF_{IO}^{\infty}(pc, F(pc)))
\end{array}
$$

Formally

$$
\begin{array}{|l}
\mathit{fun2ValueAux} : (\mathit{EVENT} \pfun \mathit{EVENT}) \pfun \mathbb{P}\ \mathit{VALUE} \\
\hline
\forall f : \mathit{EVENT} \pfun \mathit{EVENT} \bullet \\
\quad \exists e_1, e_2 : \mathit{EVENT} \\
\quad\quad | \ e_1 \in \mathit{dom}(f) \land f(e_1) = e_2 \\
\quad\quad \bullet (\# f = 1 \Rightarrow \mathit{fun2ValueAux}(f) = \{\mathit{Pair}(\mathit{Ev}(e_1), \mathit{Ev}(e_2))\}) \\
\quad\quad\quad \land (\# f > 1 \Rightarrow \mathit{fun2ValueAux}(f) = \\
\quad\quad\quad\quad \{\mathit{Pair}(\mathit{Ev}(e_1), \mathit{Ev}(e_2))\} \cup \mathit{fun2ValueAux}(\{e_1\} \ndres f))
\end{array}
$$

$$
\begin{array}{|l}
\mathit{fun2Value} : (\mathit{EVENT} \pfun \mathit{EVENT}) \pfun \mathit{VALUE} \\
\hline
\forall f : \mathit{EVENT} \pfun \mathit{EVENT} \bullet \mathit{fun2Value}(f) = \mathit{Set}(\mathit{fun2ValueAux}(f))
\end{array}
$$

$$
\begin{array}{|l}
B^{\infty} : (\mathit{EVENT} \pfun \mathit{EVENT}) \pfun \mathit{CIRCUS\_PROCESS} \\
\hline
\forall \mathit{LR} : \mathit{EVENT} \pfun \mathit{EVENT} \bullet \exists x, B, f, s : N \bullet \\
\quad B^{\infty}(\mathit{LR}) = \\
\quad\quad \mathbf{beginend}\ (\mathit{Stateless}(\\
\quad\quad \langle \mathit{ActDef}(B, \bullet_A\ (\langle (s, \mathrm{seq}(\mathbb{E})), (f, \pfun (\mathbb{E}, \mathbb{E})) \rangle, \\
\quad\quad\quad\quad \mathit{BAct}(\Box_A\ (\Box_{i\,A}(\langle (x, \mathrm{dom}(f)) \rangle, \\
\quad\quad\quad\quad\quad\quad\quad \rightarrow (\mathit{RefComm}(x), \\
\quad\quad\quad\quad\quad\quad\quad\quad \mathit{AInstArgs}(B, \langle \mathit{SeqExp}(\frown(\mathit{SeqDisplay}(\langle \mathit{Var}(x) \rangle), \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad \mathit{SeqName}(s))), \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad \mathit{Var}(f) \rangle)))), \\
\quad\quad\quad\quad\quad\quad\quad \rightarrow (\mathit{ExpComm}(\mathit{FunExp}(f, \langle \mathit{SeqExp}(\mathit{last}(\mathit{SeqName}(s))) \rangle)) \\
\quad\quad\quad\quad\quad\quad\quad\quad \mathit{AInstArgs}(B, \langle \mathit{SeqExp}(\mathit{front}(\mathit{SeqName}(s))), \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad \mathit{Var}(f) \rangle))))))) \rangle, \\
\quad\quad \mathit{AInstArgs}(B, \langle \mathit{SeqExp}(\mathit{SeqDisplay}(\langle\rangle)), \mathit{Value}(\mathit{fun2Value}(\mathit{LR})) \rangle))))
\end{array}
$$

$$
\begin{array}{|l}
B^{\infty}_{IO} : ((\mathit{CIRCUS\_PROCESS} \times \mathit{CHANNEL}) \times \\
\quad\quad (\mathit{CIRCUS\_PROCESS} \times \mathit{CHANNEL})) \pfun \mathit{CIRCUS\_PROCESS} \\
\hline
\forall p_1, p_2 : \mathit{CIRCUS\_PROCESS};\ c_1, c_2 : \mathit{CHANNEL} \bullet \\
\quad B^{\infty}_{IO}((p_1, c_1), (p_2, c_2)) = \\
\quad\quad \interleave_P\ (B^{\infty}(\mathit{BR}_{IO}((p_1, c_1), (p_2, c_2))), B^{\infty}(\mathit{BR}_{IO}((p_2, c_2), (p_1, c_1))))
\end{array}
$$

$$
\begin{array}{|l}
|||\ B_{IO}^{\infty} : ((CIRCUS\_PROCESS \times CHANNEL) \nrightarrow \\
\qquad\qquad (CIRCUS\_PROCESS \times CHANNEL)) \nrightarrow CIRCUS\_PROCESS \\
\hline
\forall f : (CIRCUS\_PROCESS \times CHANNEL) \nrightarrow \\
\quad (CIRCUS\_PROCESS \times CHANNEL) \bullet \\
\qquad \exists\, pc, fpc : CIRCUS\_PROCESS \times CHANNEL \\
\qquad \mid pc \in dom(f) \land f(pc) = fpc \\
\qquad \bullet\ (\#f = 1 \Rightarrow |||\ B_{IO}^{\infty}(f) = B_{IO}^{\infty}(pc, fpc)) \\
\qquad\quad \land\ (\#f > 1 \Rightarrow |||\ B_{IO}^{\infty}(f) = |||_P\ (B_{IO}^{\infty}(pc, fpc), |||\ B_{IO}^{\infty}(\{pc\} \ntriangleleft f)))
\end{array}
$$

$$
\begin{array}{|l}
AsyncComp_b : (CIRCUS\_PROCESS \times CIRCUS\_PROCESS \times \\
\qquad\qquad ((CIRCUS\_PROCESS \times CHANNEL) \nrightarrow \\
\qquad\qquad\quad (CIRCUS\_PROCESS \times CHANNEL))) \rightarrow \\
\qquad\qquad\qquad CIRCUS\_PROCESS \\
\hline
\forall\, p, q : CIRCUS\_PROCESS; \\
\quad f : (CIRCUS\_PROCESS \times CHANNEL) \rightarrow \\
\qquad (CIRCUS\_PROCESS \times CHANNEL) \bullet \\
\quad AsyncComp_b(p, q, f) = \\
\qquad \|_P\ (CSDisplay(channels(\alpha(p) \cup \alpha(q))), |||_P\ (p, q), |||\ B_{IO}^{\infty}(f))
\end{array}
$$

$$
\begin{array}{|l}
AsyncComp_u : (CIRCUS\_PROCESS \times \\
\qquad\qquad ((CIRCUS\_PROCESS \times CHANNEL) \nrightarrow \\
\qquad\qquad\quad (CIRCUS\_PROCESS \times CHANNEL))) \rightarrow \\
\qquad\qquad\qquad CIRCUS\_PROCESS \\
\hline
\forall\, p : CIRCUS\_PROCESS; \\
\quad f : (CIRCUS\_PROCESS \times CHANNEL) \rightarrow \\
\qquad (CIRCUS\_PROCESS \times CHANNEL) \bullet \\
\quad AsyncComp_u(p, f) = \\
\qquad \|_P\ (CSDisplay(channels(\alpha(p))), p, |||\ B_{IO}^{\infty}(f))
\end{array}
$$

Definition I.3 (Asynchronous binary composition) *Let P and Q be two distinct component contracts, and $\langle c_1, .., c_n\rangle$ and $\langle z_1, .., z_n\rangle$ sequences of distinct channels within $\mathcal{C}(P)$ and $\mathcal{C}(Q)$, respectively, such that $\mathcal{C}(P) \cap \mathcal{C}(Q) = \emptyset$. Then, the asynchronous binary composition of P and Q (namely $P_{\langle c_1,..,c_n\rangle} \asymp {}_{\langle z_1,..,z_n\rangle} Q$) is given by:*

$$
\begin{array}{l}
P_{\langle c_1,..,c_n\rangle} \asymp {}_{\langle z_1,..,z_n\rangle} Q = \\
\quad \langle AsyncComp(\{\mathcal{B}(P), \mathcal{B}(Q)\}, \{c_i \mapsto z_i \mid i \in 1..n\}), \mathcal{R}(PQ), \mathcal{I}(PQ), \mathcal{C}(PQ)\rangle
\end{array}
$$

where

- $\mathcal{C}(PQ) = (\mathcal{C}(P) \cup \mathcal{C}(Q)) \setminus \{c_1, .., c_n, z_1, .., z_n\}$
- $\mathcal{R}(PQ) = \mathcal{C}(PQ) \lhd (\mathcal{R}(P) \cup \mathcal{R}(Q))$
- $\mathcal{I}(PQ) = \operatorname{ran} \mathcal{R}(PQ)$

Formally

function 30 leftassoc(_ [_ $\asymp$ _] _)

```
_ [ _ ≍ _ ] _ : $CTR \times (\text{iseq}\, CHANNEL) \times (\text{iseq}\, CHANNEL) \times CTR \nrightarrow CTR$
──────────────
$\forall P, Q : CTR;\ s, t : \text{iseq}\, CHANNEL$
    $\mid \#s = \#t \land \operatorname{ran} s \subseteq \mathcal{C}(P) \land \operatorname{ran} t \subseteq \mathcal{C}(Q) \land \operatorname{ran} s \cap \operatorname{ran} t = \emptyset$
    $\bullet \exists BricCPQ : \mathbb{P}\, CHANNEL$
        $\mid BricCPQ = (\mathcal{C}(P) \cup \mathcal{C}(Q)) \setminus (\operatorname{ran} s \cup \operatorname{ran} t)$
        $\bullet \exists BricRPQ : (CHANNEL \rightarrow TYPE)$
            $\mid BricRPQ = BricCPQ \lhd (\mathcal{R}(P) \cup \mathcal{R}(Q))$
            $\bullet P\, [\, s \asymp t\, ]\, Q =$
                $(AsyncComp_b\, (\mathcal{B}(P), \mathcal{B}(Q),$
                    $\{i : 0 \mathrel{..} (\#s) \bullet (\mathcal{B}(P), s(i)) \mapsto (\mathcal{B}(P), t(i))\}),$
                $BricRPQ,$
                $\operatorname{ran} BricRPQ,$
                $BricCPQ)$
```

Notice that we have injective sequences in the function domain. Furthermore, we have changed the definition of *AsyncComp* for the reasons we explain below.

I.2.7 Asynchronous Unary Composition

Definition I.4 (Asynchronous unary composition) *Let P be a component contract, and $\langle c_1, .., c_n\rangle$ and $\langle z_1, .., z_n\rangle$ sequences of distinct channels within $\mathcal{C}(P)$, such that $\{c_1, .., c_n\} \cap \{z_1, .., z_n\} = \emptyset$. Then, the asynchronous unary composition of P (namely $P \asymp|^{\langle c_1,..,c_n\rangle}_{\langle z_1,..,z_n\rangle}$) is given by:*

$$P \asymp|^{\langle c_1,..,c_n\rangle}_{\langle z_1,..,z_n\rangle} = \langle (AsyncComp(\{\mathcal{B}(P)\}, \{c_i \mapsto z_i \mid i \in 1..n\}), \mathcal{R}(PQ), \mathcal{I}(PQ), \mathcal{C}(PQ)\rangle$$

where

- $\mathcal{C}(PQ) = \mathcal{C}(P) \setminus \{c_1, .., c_n, z_1, .., z_n\}$
- $\mathcal{R}(PQ) = \mathcal{C}(PQ) \lhd \mathcal{R}(P)$
- $\mathcal{I}(PQ) = \operatorname{ran} \mathcal{R}(PQ)$

Formally

$$\text{function } 30 \text{ leftassoc}(\_ \asymp\!| \_)$$

```
$\_ \asymp\!| \_ : (CTR \times (\operatorname{iseq} CHANNEL \times \operatorname{iseq} CHANNEL)) \nrightarrow CTR$
----
$\forall P : CTR;\ s, t : \operatorname{seq} CHANNEL$
    $| \#s = \#t \land \operatorname{ran}(s) \cup \operatorname{ran}(t) \subseteq \mathcal{C}(P) \land \operatorname{ran} s \cap \operatorname{ran} t = \emptyset$
    $\bullet \exists BricCPQ : \mathbb{P}\, CHANNEL$
        $| BricCPQ = \mathcal{C}(P) \setminus (\operatorname{ran} s \cup \operatorname{ran} t)$
        $\bullet \exists BricRPQ : (CHANNEL \rightarrow TYPE)$
            $| BricRPQ = BricCPQ \lhd \mathcal{R}(P)$
            $\bullet P \asymp\!| (s, t) =$
                $(AsyncComp_u\, (\mathcal{B}(P), \{i : 0 \,..\, (\#s) \bullet (\mathcal{B}(P), s(i)) \mapsto (\mathcal{B}(P), t(i))\}),$
                    $BricRPQ,$
                    $\operatorname{ran} BricRPQ,$
                    $BricCPQ)$
```

I.2.8 Projection

New Definition I.1 (Projection) *Let P be an I/O Process, and C a set of communication channels. The projection of P over C (denoted by $P \upharpoonright C$) satisfies the following properties:*

1. *$P \upharpoonright C$ is an I/O Process*
2. *$\forall c : C \bullet inputs(P \upharpoonright C, c) \subseteq inputs(P, c)$*
3. *$\forall c : C \bullet outputs(P \upharpoonright C, c) \subseteq outputs(P, c)$*
   - *Test characterisation: as before*
4. *$\alpha(P \upharpoonright C) \subseteq \bigcup_{c:C} \{\!| c |\!\}$*
   - *Test characterisation: $ProtCheck(P \upharpoonright C, C)$ is deadlock-free*

$$ProtCheck(P, C) = P \,[\![\, NOT(C) \,]\!]\, PRUNE(NOT(C))$$
$$PRUNE(A) = \Box\, ev : A \bullet ev \rightarrow Stop$$
$$NOT(C) = \Sigma \setminus \bigcup_{c:C} \{\!| c |\!\}$$

5. *$P \equiv_{\mathrm{T}} P \,[\![\, \Sigma \,]\!]\, ((P \upharpoonright C) \,|||\, RUN(NOT(C)))$*

$$RUN(CS) = \Box\, c : CS \bullet c \rightarrow RUN(CS)$$

Formally

```
| $\Box : \mathbb{F}_1\, EVENT \rightarrow ActBody$
|---
| $\forall es : \mathbb{P}_1\, EVENT \bullet$
|     $\exists e : es \bullet$
|         $\#es = 1 \Rightarrow \Box(es) =$
|             $\rightarrow (BasicComm(e.1, \langle syncc(Value(e.2))\rangle), Skip)$
|         $\wedge\ \#es > 1 \Rightarrow \Box(es) =$
|             $\Box_A\, (\rightarrow (BasicComm(e.1, \langle syncc(Value(e.2))\rangle), Skip), \Box(es \setminus \{e\}))$
```

```
| $RUN : \mathbb{F}_1\, CHANNEL \rightarrow CIRCUS\_PROCESS$
|---
| $\forall cs : \mathbb{F}_1\, CHANNEL \bullet$
|     $\exists X : N \bullet$
|         $RUN(cs) =$
|             $\mathbf{beginend}\ (Stateless(\langle\rangle, \mu(X, ;_A(\Box(\{\!|\ cs\ |\!\}), AInst(X)))))$
```

function 30 leftassoc$(_ \restriction_B _)$

$$\begin{array}{|l}
\_ \restriction_B \_ : \mathit{CIRCUS\_PROCESS} \times \mathbb{P}\,\mathit{CHANNEL} \rightarrow \mathit{CIRCUS\_PROCESS} \\
\hline
\forall p : \mathit{CIRCUS\_PROCESS};\ cs : \mathbb{P}\,\mathit{CHANNEL} \bullet \\
\quad p \restriction_B cs \in \mathit{IOProcess} \\
\quad \land \forall c : cs \bullet \mathit{inputs}(c, p \restriction_B cs) \subseteq \mathit{inputs}(c, p) \\
\quad \land \forall c : cs \bullet \mathit{outputs}(c, p \restriction_B cs) \subseteq \mathit{outputs}(c, p) \\
\quad \land \alpha(p \restriction_B cs) \subseteq \{\!| \mid cs \mid |\!\} \\
\quad \land p \equiv_{\mathrm{T}} \|_P\ (\mathit{CSDisplay}(\mathit{CHANNEL}), p, |||_P\ (p \restriction_B cs, \mathit{RUN}(\mathit{CHANNEL} \setminus cs)))
\end{array}$$

I.2.9 Communication protocol

Definition E.2 (Communication protocol) *We say a* ***Circus*** *process P is a communication protocol if:*

- $\exists c_1, c_2 \bullet \mathit{inputs}(P) \subseteq \{\!|\ c_1\ |\!\} \land \mathit{outputs}(P) \subseteq \{\!|\ c_2\ |\!\}$;

Formally

$$\begin{array}{|l}
\mathit{CommProt} : \mathbb{P}(\mathit{CIRCUS\_PROCESS}) \\
\hline
\forall P : \mathit{CIRCUS\_PROCESS} \bullet \\
\quad P \in \mathit{CommProt} \\
\quad \Leftrightarrow \\
\quad \exists c_1, c_2 : \mathit{CHANNEL} \bullet \mathit{inputs}_P(P) \subseteq \{\!|\ c_1\ |\!\} \land \mathit{outputs}_P(P) \subseteq \{\!|\ c_2\ |\!\}
\end{array}$$

I.2.10 Protocol Implementation

New Definition 4.1 (Protocol implementation) *Let P be an I/O process, and ch a communication channel. The communication protocol, namely $\mathit{Prot}_{\mathit{IMP}}(P, ch)$, implemented by P over ch is a protocol that satisfies the following property:*

$$\mathit{Prot}_{\mathit{IMP}}(P, ch) \equiv_{\mathrm{F}} P \restriction ch$$

Formally

$$
\begin{array}{|l}
Prot_{IMP} : (CIRCUS\_PROCESS \times CHANNEL) \rightarrow CIRCUS\_PROCESS \\
\hline
\forall P : CIRCUS\_PROCESS;\ c : CHANNEL \bullet \\
\quad Prot_{IMP}(P, c) = P \upharpoonright_B \{c\} \\
\quad \land\ Prot_{IMP}(P, c) \in CommProt
\end{array}
$$

I.2.11 Dual Protocol

Definition E.3 (Dual protocol) *Let P be a deadlock-free communication protocol. The dual protocol of P is defined as a deadlock-free communication protocol DP, such that:*

$$
\begin{array}{l}
inputs(P) = outputs(DP) \\
\land\ outputs(P) = inputs(DP) \\
\land\ traces(DP) = traces(P)
\end{array}
$$

Formally

$$
\begin{array}{|l}
DP : CIRCUS\_PROCESS \rightarrow CIRCUS\_PROCESS \\
\hline
\forall P : CIRCUS\_PROCESS \\
\quad \mid P \in CommProt \land P \in DeadlockFree \\
\quad \bullet\ inputs_P(P) = outputs_P(DP(P)) \\
\qquad \land\ outputs_P(P) = inputs_P(DP(P)) \\
\qquad \land\ traces(P) = traces(DP(P))
\end{array}
$$

I.2.12 Dual Protocol

Definition I.5 (Communication context process) *Let P be a deadlock-free communication protocol. The communication context process of P (denoted by CTX_P) is defined as a deadlock-free deterministic process, such that $traces(CTX_P) = traces(P)$.*

Formally

$$
\begin{array}{|l}
CTX : CIRCUS\_PROCESS \rightarrow CIRCUS\_PROCESS \\
\hline
\forall P : CIRCUS\_PROCESS \\
\quad \mid P \in CommProt \land P \in DeadlockFree \\
\quad \bullet\; CTX(P) \in DeadlockFree \\
\qquad \land\; CTX(P) \in Deterministic \\
\qquad \land\; traces(P) = traces(CTX(P))
\end{array}
$$

I.2.13 Renaming I/O

$$
\begin{array}{|l}
R_{IO} : (CIRCUS\_PROCESS \times CHANNEL \times CHANNEL) \rightarrow CIRCUS\_PROCESS \\
\hline
\forall P : CIRCUS\_PROCESS;\; a, b : CHANNEL \\
\quad \bullet\; \exists f : (EVENT \rightarrow EVENT) \\
\quad \mid f = BR_{IO}((P, a), (P, b)) \\
\quad \bullet\; traces(R_{IO}\,(P, a, b)) = replace_t(traces(P), f) \\
\qquad \land\; divergences(R_{IO}\,(P, a, b)) = replace_t(divergences(P), f) \\
\qquad \land\; failures(R_{IO}\,(P, a, b)) = replace_f(failures(P), f)
\end{array}
$$

$$
\begin{array}{|l}
R_{IMP} : (CIRCUS\_PROCESS \times CHANNEL \times CHANNEL) \rightarrow CIRCUS\_PROCESS \\
\hline
\forall P : CIRCUS\_PROCESS;\; a, b : CHANNEL \\
\quad \bullet\; R_{IMP}(P, a, b) = R_{IO}(Prot_{IMP}(P, a), a, b)
\end{array}
$$

I.2.14 I/O confluence

Definition I.6 (I/O confluence) *Let P be an I/O process. Then P is I/O confluent if and only if :*

$$
\begin{array}{l}
\forall\, s \frown \langle c_1.a\rangle \frown t, s \frown \langle c_2.b\rangle : traces(P) \;\mid\; c_1.a \neq c_2.b \bullet \\
\quad (c_1.a \in inputs(P) \land \\
\qquad \exists\, i : inputs(P, c_1) \;\mid\; s \frown \langle c_2.b, i\rangle \frown (t - \langle c_2.b\rangle) \in traces(P)) \\
\quad \lor\, (c_1.a \in outputs(P) \land \\
\qquad \exists\, o : outputs(P, c_1) \;\mid\; s \frown \langle c_2.b, o\rangle \frown (t - \langle c_2.b\rangle) \in traces(P)) \\
\quad \lor\, (c_1 = c_2 \land (\{c_1.a, c_2.b\} \subseteq outputs(P) \lor \{c_1.a, c_2.b\} \subseteq inputs(P)))
\end{array}
$$

Formally

$$\begin{array}{|l}
\mathit{IOConfluent} : \mathbb{P}\,\mathit{CIRCUS\_PROCESS} \\
\hline
\forall\, p : \mathit{CIRCUS\_PROCESS} \bullet \\
\quad p \in \mathit{IOConfluent} \\
\quad \Leftrightarrow \\
\quad (p \in \mathit{IOProcess} \\
\quad \wedge\ (\forall\, e_1, e_2 : \mathit{EVENT} \\
\qquad \mid e_1 \neq e_2 \\
\qquad \bullet \forall\, s, t : \mathit{TRACE} \\
\qquad\quad \mid \{s \frown \langle e_1 \rangle \frown t, s \frown \langle e_2 \rangle\} \subseteq \mathit{traces}(p) \\
\qquad\quad \bullet (e_1 \in \mathit{inputs}_P(p) \\
\qquad\qquad \wedge\ \exists\, i : \mathit{inputs}(\mathit{channel}(e_1), p) \\
\qquad\qquad\quad \bullet\ s \frown \langle e_2, i \rangle \frown (t -_m \langle e_2 \rangle) \in \mathit{traces}(p)) \\
\qquad\quad \vee\ (e_1 \in \mathit{outputs}_P(p) \\
\qquad\qquad \wedge\ \exists\, o : \mathit{outputs}(\mathit{channel}(e_1), p) \\
\qquad\qquad\quad \bullet\ s \frown \langle e_2, o \rangle \frown (t -_m \langle e_2 \rangle) \in \mathit{traces}(p)) \\
\qquad\quad \vee\ (\mathit{channel}(e_1) = \mathit{channel}(e_2) \\
\qquad\qquad \wedge\ \{e_1, e_2\} \subseteq \mathit{outputs}_P(p) \vee \{e_1, e_2\} \subseteq \mathit{inputs}_P(p)))\,)
\end{array}$$

I.2.15 Conjugate protocols

Definition I.7 (Conjugate protocols) *Let P and Q be two communication protocols. P and Q are conjugate if, and only if:*

- $\mathit{outputs}(P) \subseteq \mathit{inputs}(Q) \wedge \mathit{outputs}(Q) \subseteq \mathit{inputs}(P)$
- $\mathit{outputs}(P) \cap \mathit{outputs}(Q) = \emptyset \wedge \mathit{inputs}(P) \cap \mathit{inputs}(Q) = \emptyset$

Formally

$$\begin{array}{|l}
\mathit{ConjugateProtocols} : \mathit{CIRCUS\_PROCESS} \leftrightarrow \mathit{CIRCUS\_PROCESS} \\
\hline
\forall\, p, q : \mathit{CIRCUS\_PROCESS} \bullet \\
\quad (p, q) \in \mathit{ConjugateProtocols} \\
\quad \Leftrightarrow \\
\quad (\mathit{outputs}_P(p) \subseteq \mathit{inputs}_P(q) \\
\qquad \wedge\ \mathit{outputs}_P(q) \subseteq \mathit{inputs}_P(p) \\
\qquad \wedge\ \mathit{outputs}_P(p) \cap \mathit{outputs}_P(q) = \emptyset \\
\qquad \wedge\ \mathit{inputs}_P(p) \cap \mathit{inputs}_P(q) = \emptyset)
\end{array}$$

I.2.16 Strong protocol compatibility

Definition I.8 (Strong protocol compatibility) *Let P and Q be two deadlock-free communication protocols, such that P and Q are conjugate. The protocols P and Q are strong compatible (denoted $P \tilde{\approx} Q$) if, and only if:*

$$\forall s : traces(P) \cap traces(Q) \bullet (\ O_P^s \neq \emptyset \lor \ O_Q^s \neq \emptyset) \land \ O_P^s \subseteq I_Q^s \land \ O_Q^s \subseteq I_P^s$$

$$I_P^s = \{a : inputs(P) \mid s \frown \langle a \rangle \in traces(P)\}$$
$$O_P^s = \{a : outputs(P) \mid s \frown \langle a \rangle \in traces(P)\}$$

Formally

$$\begin{array}{|l}
I, O : TRACE \times CIRCUS\_PROCESS \rightarrow \mathbb{P}\, EVENT \\
\hline
\forall s : TRACE;\ p : CIRCUS\_PROCESS \bullet \\
\quad I\,(s, p) = \{a : inputs_P(p) \mid s \frown \langle a \rangle \in traces(p)\} \\
\quad \land\ O\,(s, p) = \{a : outputs_P(p) \mid s \frown \langle a \rangle \in traces(p)\}
\end{array}$$

relation($\_ \tilde{\approx} \_$)

$$\begin{array}{|l}
\_ \tilde{\approx} \_ : CIRCUS\_PROCESS \leftrightarrow CIRCUS\_PROCESS \\
\hline
\forall p, q : CIRCUS\_PROCESS \bullet \\
\quad p \tilde{\approx} q \\
\quad \Leftrightarrow \\
\quad (\ \{p, q\} \subseteq DeadlockFree \\
\qquad \land\ (p, q) \in ConjugateProtocols \\
\qquad \land\ \forall s : traces(p) \cap traces(q) \\
\qquad\quad \bullet\ (O(s, p) \neq \emptyset \lor O(s, q) \neq \emptyset) \\
\qquad\qquad \land\ O(s, p) \subseteq I(s, q) \\
\qquad\qquad \land\ O(s, q) \subseteq I(s, p)\ )
\end{array}$$

I.2.17 Finite output property

Definition I.9 (Finite output property) *Let P be an I/O process, and C the set of channels used in P. P satisfies the finite output property (FOP) if, and only if, for all $c \in C$ the process $P \setminus outputs(P, c)$ is divergence-free.*

Formally

$$
\begin{array}{|l}
FOP : \mathbb{P}\, CIRCUS\_PROCESS \\
\hline
\forall\, p : CIRCUS\_PROCESS \bullet \\
\quad p \in FOP \\
\quad \Leftrightarrow \\
\quad (\, p \in IOProcess \land \\
\qquad \setminus_P\ (CSDisplay(channels(outputs_P(p))), p) \in DivergenceFree\,)
\end{array}
$$

I.2.18 Decoupled Channels

Definition I.10 (Decoupled channels) *Let P be an I/O process and Ch a set of channels. Then, the channels within Ch are decoupled in P (denoted by Ch DecoupledIn P) if, and only, if:*

$$P \upharpoonright Ch \equiv_{\mathrm{F}} \interleave_{z \in cs} Prot_{IMP}(p, z)$$

Formally

relation($\_\, DecoupledIn\, \_$)

$$
\begin{array}{|l}
\interleave\ Prot_{IMP} : (CIRCUS\_PROCESS \times \mathbb{P}_1\, CHANNEL) \rightarrow CIRCUS\_PROCESS \\
\hline
\forall\, p : CIRCUS\_PROCESS;\ cs : \mathbb{P}_1\, CHANNEL \bullet \\
\quad \exists\, c : cs \bullet \\
\qquad \#cs = 1 \Rightarrow \interleave\ Prot_{IMP}(p, cs) = Prot_{IMP}(p, c) \\
\qquad \land\ \#cs > 1 \Rightarrow \interleave\ Prot_{IMP}(p, cs) = \interleave_P\ (Prot_{IMP}(p, c), \interleave\ Prot_{IMP}(p, cs \setminus \{c\}))
\end{array}
$$

$$
\begin{array}{|l}
\_\ DecoupledIn\ \_ : \mathbb{P}\, CHANNEL \leftrightarrow CIRCUS\_PROCESS \\
\hline
\forall\, cs : \mathbb{P}\, CHANNEL;\ p : CIRCUS\_PROCESS \bullet \\
\quad cs\ decoupled\ p \\
\quad \Leftrightarrow \\
\quad (p \in IOProcess \land p \upharpoonright_B cs \equiv_{\mathrm{F}} \interleave\ Prot_{IMP}(p, cs))
\end{array}
$$

I.2.19 Buffering self-injection compatibility

Definition I.11 (Buffering self-injection compatibility) *Let P be a deadlock-free I/O process, and c and z channels. Then $Pj = P \upharpoonright \{c, z\}$ is buffering self-injection compatible if, and only if:*

1. $\forall (s, X) : failures(Pj) \mid (s \downarrow O_c = s \downarrow I_z) \land (s \downarrow O_z = s \downarrow I_c) \bullet X \cap (O_c \cup O_z) = \emptyset$
2. $\forall (s, X) : failures(Pj) \mid s \downarrow O_c > s \downarrow I_z \bullet (s \upharpoonright z, X \cup \{c\}) \in failures(Pj \upharpoonright z)$
3. $\forall (s, X) : failures(Pj) \mid s \downarrow O_z > s \downarrow I_c \bullet (s \upharpoonright c, X \cup \{z\}) \in failures(Pj \upharpoonright c)$

where $O_c = outputs(P, c)$, $O_z = outputs(P, z)$, $I_c = inputs(P, c)$ *and* $I_z = inputs(P, z)$

relation($\_\, buffSelfInjComp\, \_$)

$$
\begin{array}{|l}
\_\, buffSelfInjComp\, \_ : CIRCUS\_PROCESS \leftrightarrow (CHANNEL \times CHANNEL) \\
\hline
\forall p : CIRCUS\_PROCESS;\ c, z : CHANNEL \bullet \\
\quad p\ buffSelfInjComp\ (c, z) \\
\quad \Leftrightarrow \\
\quad (\, p \in IOProcess \land p \in DeadlockFree \\
\quad \land \forall t : TRACE;\ r : REFUSAL \mid (t, r) \in failures(p \upharpoonright_B \{c, z\}) \bullet \\
\qquad ((t \downarrow_S outputs(c, p) = t \downarrow_S inputs(z, p) \\
\qquad\quad \land\ t \downarrow_S outputs(z, p) = t \downarrow_S inputs(c, p)) \Rightarrow \\
\qquad\qquad r \cap (outputs(c, p) \cup outputs(z, p)) = \emptyset) \\
\qquad \land\ (t \downarrow_S outputs(c, p) > t \downarrow_S inputs(z, p) \Rightarrow \\
\qquad\qquad (t \upharpoonright \{\!|\ z\ |\!\}, r \cup \{\!|\ c\ |\!\}) \in failures((p \upharpoonright_B \{c, z\}) \upharpoonright_B \{z\})) \\
\qquad \land\ (t \downarrow_S outputs(z, p) > t \downarrow_S inputs(c, p) \Rightarrow \\
\qquad\qquad (t \upharpoonright \{\!|\ c\ |\!\}, r \cup \{\!|\ z\ |\!\}) \in failures((p \upharpoonright_B \{c, z\}) \upharpoonright_B \{c\}))\,)
\end{array}
$$

I.2.20 Interaction patterns

Definition I.12 (Interaction patterns) *Let* P *be a CSP process.*

$$InteractionPatterns(P) = \{s : traces(P) \mid P \sqsubseteq_{\mathrm{FD}} (P/s) \land s \textit{ is finite}\}$$

Formally

$$
\begin{array}{|l}
InteractionPatterns : CIRCUS\_PROCESS \rightarrow\!\!\!\!\!+\ \mathbb{P}\, TRACE \\
\hline
\forall P : CIRCUS\_PROCESS \\
\quad \bullet\ InteractionPatterns(P) = \\
\qquad \{s : TRACE \mid s \in traces(P) \land P \sqsubseteq_{\mathrm{FD}} (P/s)\}
\end{array}
$$

I.2.21 Interaction process

Definition I.13 (Interaction process) *A divergence-free CSP process P is an interaction process if, and only if:*

$$\forall s \in traces(P) \bullet \exists p : InteractionPatterns(P) \bullet s \preceq p$$

Formally

$$\begin{array}{|l}
InteractionProcess : \mathbb{P}\, CIRCUS\_PROCESS \\
\hline
\forall p : CIRCUS\_PROCESS \\
\quad \bullet\ p \in InteractionProcess \\
\quad \Leftrightarrow \\
\quad \forall s : traces(p) \bullet \exists t : InteractionPatterns(p) \bullet s \mathrel{prefix} t
\end{array}$$

I.2.22 Interaction component

Definition I.14 (Interaction component) *Let C be a component with contract Ctr. Then C is an interaction component if, and only if, $\mathcal{B}_{Ctr}$ is an interaction process.*

Formally

$$\begin{array}{|l}
InteractionComponent : \mathbb{P}\, CTR \\
\hline
\forall c : CTR \\
\quad \bullet\ c \in InteractionComponent \\
\quad \Leftrightarrow \\
\quad \mathcal{B}(c) \in InteractionProcess
\end{array}$$

I.2.23 Interaction channels

Definition I.15 (Interaction channels) *Let Ctr be an interaction component contract. Then its interaction channels are:*

$$IntCh_{Ctr} = \{chans(t) \mid t \in InteractionPatterns(\mathcal{B}_{Ctr})\}$$

Formally

$$\begin{array}{|l}
\mathit{IntCh} : \mathit{CTR} \rightarrow \mathbb{P}\,\mathit{CHANNEL} \\
\hline
\forall\, c : \mathit{CTR} \\
\quad\quad \bullet\ \mathit{IntCh}(c) = \bigcup\{t : \mathit{InteractionPatterns}(\mathcal{B}(c)) \bullet \mathit{channels}(\operatorname{ran}(t))\}
\end{array}$$

I.2.24 Wrapping

Definition I.16 (Wrapping) *Let P be a component contract, and CC a set of channels. Then, a wrapping of P with respect to CC is given by:*

$P \setminus CC = \langle (\mathcal{B}_P \setminus \{\mid CC \ \setminus\ \mathcal{C}_P \mid\}, \mathcal{R}_P, \mathcal{I}_P, \mathcal{C}_P \rangle$

Formally

function 30 leftassoc(_ ⌈⌈ _)

$$\begin{array}{|l}
\_ \lceil\lceil \_ : \mathit{CTR} \times \mathbb{P}\,\mathit{CHANNEL} \nrightarrow \mathit{CTR} \\
\hline
\forall\, c : \mathit{CTR};\ \mathit{CC} : \mathbb{P}\,\mathit{CHANNEL} \bullet \\
\quad\quad c \lceil\lceil \mathit{CC} = (\setminus_P\ (\mathit{CSDisplay}(\mathit{channels}(\{\!|\mid \mathit{CC} \setminus \mathcal{C}(c) \mid |\!\})), \mathcal{B}(c)), \\
\quad\quad\quad\quad\quad\quad\quad\quad \mathcal{R}(c), \mathcal{I}(c), \mathcal{C}(c))
\end{array}$$

I.3 Composition Rules

I.3.1 Interleave Composition

Definition I.17 (Interleave composition) *Let P and Q be two component contracts, such that P and Q have disjoint channels, $\mathcal{C}_P \cap \mathcal{C}_Q = \emptyset$. Then, the interleave composition of P and Q (namely $P\,[|||]\,Q$) is given by:*

$P\,[|||]\,Q = P\ {}_{\langle\rangle} \asymp {}_{\langle\rangle}\ Q$

Formally

function 30 leftassoc(_ [|||] _)

$$\begin{array}{|l}
\_ [|||] \_ : (\mathit{CTR} \times \mathit{CTR}) \nrightarrow \mathit{CTR} \\
\hline
\forall\, P, Q : \mathit{CTR} \\
\quad\quad \mid \mathcal{C}(P) \cap \mathcal{C}(Q) = \emptyset \\
\quad\quad \bullet\ P\,[|||]\,Q = P\,[\,\langle\rangle \asymp \langle\rangle\,]\,Q
\end{array}$$

I.3.2 Communication Composition

Definition 3.3 (Communication composition) *Let P and Q be two component contracts, and ic and oc two communication channels, such that $ic \in \mathcal{C}_P \wedge oc \in \mathcal{C}_Q$, $\mathcal{C}_P \cap \mathcal{C}_Q = \emptyset$, and the port-protocols $Prot_{IMP}(P, ic) \llbracket R_{IO}^{ic \to oc} \rrbracket$ and $Prot_{IMP}(Q, oc) \llbracket R_{IO}^{oc \to ic} \rrbracket$ are I/O confluent strong compatible and satisfy the finite output property. Then, the communication composition of P and Q (namely $P[ic \leftrightarrow oc]Q$) via ic and oc is defined as follows:*

$$P[ic \leftrightarrow oc]Q = P_{\langle ic \rangle} \asymp {}_{\langle oc \rangle} Q$$

Formally

function 30 leftassoc$(_ [_ \leftrightarrow _] _)$

$$
\begin{array}{|l}
\_ [\_ \leftrightarrow \_] \_ : CTR \times CHANNEL \times CHANNEL \times CTR \nrightarrow CTR \\
\hline
\forall P, Q : CTR;\ ic, oc : CHANNEL \\
\quad \mid (\, ic \in \mathcal{C}(P) \wedge oc \in \mathcal{C}(Q) \\
\qquad \wedge\, \mathcal{C}(P) \cap \mathcal{C}(Q) = \emptyset \\
\qquad \wedge\, \{R_{IMP}(P.1, ic, oc), R_{IMP}(Q.1, oc, ic)\} \subseteq IOConfluent \\
\qquad \wedge\, \{R_{IMP}(P.1, ic, oc), R_{IMP}(Q.1, oc, ic)\} \subseteq FOP \\
\qquad \wedge\, R_{IMP}(P.1, ic, oc) \approxeq R_{IMP}(Q.1, oc, ic)\,) \\
\quad \bullet\ P\,[\, ic \leftrightarrow oc\,]\, Q = P\,[\, \langle ic \rangle \asymp \langle oc \rangle\,]\, Q
\end{array}
$$

I.3.3 Feedback Composition

Definition 3.4 (Feedback composition) *Let P be a component contract, and ic and oc two communication channels, such that $Prot_{IMP}(P, ic) \llbracket R_{IO}^{ic \to oc} \rrbracket$ and the protocols $Prot_{IMP}(P, oc) \llbracket R_{IO}^{oc \to ic} \rrbracket$ are I/O confluent strong compatible and satisfy the finite output property, $\{ic, oc\} \subseteq \mathcal{C}_P$ and decoupled in P. Then, the feedback composition P (namely $P[oc \hookrightarrow ic]$) hooking oc to ic is defined as follows:*

$$P[oc \hookrightarrow ic] = P \asymp|_{\langle oc \rangle}^{\langle ic \rangle}$$

Formally

function$(_ [_ \hookrightarrow _])$

$$
\begin{array}{|l}
\_[\_ \hookrightarrow \_] : CTR \times CHANNEL \times CHANNEL \nrightarrow CTR \\
\hline
\forall P, Q : CTR;\ ic, oc : CHANNEL \\
\quad \mid (\, ic \neq oc \wedge \{ic, oc\} \subseteq \mathcal{C}(P) \\
\qquad \wedge \{oc, ic\}\ DecoupledIn\ P.1 \\
\qquad \wedge \{R_{IMP}(P.1, ic, oc), R_{IMP}(P.1, oc, ic)\} \subseteq IOConfluent \\
\qquad \wedge \{R_{IMP}(P.1, ic, oc), R_{IMP}(P.1, oc, ic)\} \subseteq FOP \\
\qquad \wedge R_{IMP}(P.1, ic, oc) \approx R_{IMP}(P.1, oc, ic)\,) \\
\quad \bullet P\,[\,oc \hookrightarrow ic\,] = P \asymp\!|\ (\langle oc\rangle, \langle ic\rangle)
\end{array}
$$

I.3.4 Reflexive Composition

Definition 3.5 (Reflexive composition) *Let P be a component contract, and ic and oc two communication channels, such that $\{ic, oc\} \subseteq \mathcal{C}(P)$, and $P \upharpoonright \{c, z\}$ buffering self-injection compatible and satisfies the finite output property. Then, the reflexive composition P (namely $P[oc \overset{-}{\hookrightarrow} ic]$) hooking oc to ic is defined as follows:*

$$P[ic \overset{-}{\hookrightarrow} oc] = P \asymp\!|^{\langle ic\rangle}_{\langle oc\rangle}$$

Formally

function($\_\,[\_ \overset{-}{\hookrightarrow} \_\,]$)

$$
\begin{array}{|l}
\_[\_ \overset{-}{\hookrightarrow} \_] : CTR \times CHANNEL \times CHANNEL \nrightarrow CTR \\
\hline
\forall P : CTR;\ ic, oc : CHANNEL \\
\quad \mid (\, \{ic, oc\} \subseteq \mathcal{C}(P) \\
\qquad \wedge P.1\ buffSelfInjComp\,(ic, oc) \\
\qquad \wedge P.1 \upharpoonright_B \{ic, oc\} \in FOP) \\
\quad \bullet P\,[\,ic \overset{-}{\hookrightarrow} oc\,] = P \asymp\!|\ (\langle ic\rangle, \langle oc\rangle)
\end{array}
$$

I.3.5 Extended Communication Composition

Definition I.18 (Extended communication composition) *Let P and Q be two component contracts, and $chseq_P$ and $chseq_Q$ two nonempty sequences of distinct communication channels, such that:*

- $\mathcal{C}_P \cap \mathcal{C}_Q = \emptyset \wedge \operatorname{ran} chseq_P \subseteq \mathcal{C}_P \wedge \operatorname{ran} chseq_Q \subseteq \mathcal{C}_Q$*;*

- $\mathrm{ran}\, chseq_P$ *DecoupledIn* $P \wedge \mathrm{ran}\, chseq_Q$ *DecoupledIn* $Q \wedge \# chseq_P = \# chseq_Q$*;*
- $\forall\, i, z, c \mid c = chseq_P(i) \wedge z = chseq_Q(i)$ $Prot_{IMP}(P, c) \llbracket R_{IO}^{c \to z} \rrbracket$ *and* $Prot_{IMP}(Q, z) \llbracket R_{IO}^{z \to c} \rrbracket$ *are I/O confluent strong compatible port-protocols and satisfy the finite output property;*

Then, the extended communication composition of P and Q (namely $P[chseq_P \leftrightarrow chseq_Q]Q$*) via the channels within* $chseq_P$ *and* $chseq_Q$ *is defined as follows:*

$$P[chseq_P \leftrightarrow chseq_Q]Q = P_{\,chseq_P} \asymp_{\,chseq_Q} Q$$

Formally

function 30 leftassoc$(_\,[_ \leftrightarrow_+ _\,]\,_)$

$$\_\,[\_ \leftrightarrow_+ \_\,]\,\_ : CTR \times \mathrm{iseq}\, CHANNEL \times \mathrm{iseq}\, CHANNEL \times CTR \nrightarrow CTR$$

$$\begin{array}{l}
\forall\, P, Q : CTR;\ ics, ocs : \mathrm{iseq}\, CHANNEL \\
\quad \mid (\,\mathrm{ran}\, ics \subseteq \mathcal{C}(P) \wedge \mathrm{ran}\, ocs \subseteq \mathcal{C}(Q) \\
\qquad \wedge\, \mathcal{C}(P) \cap \mathcal{C}(Q) = \emptyset \\
\qquad \wedge\, \mathrm{ran}\, ics\ \mathit{DecoupledIn}\ P.1 \\
\qquad \wedge\, \mathrm{ran}\, ocs\ \mathit{DecoupledIn}\ Q.1 \\
\qquad \wedge\, \# ics = \# ocs \\
\qquad \forall\, i : \mathrm{dom}\, ics;\ ic, oc : CHANNEL \\
\qquad\quad \mid ic = ics(i) \wedge oc = ocs(i) \\
\qquad\quad \bullet \{R_{IMP}(P.1, ic, oc), R_{IMP}(Q.1, oc, ic)\} \subseteq IOConfluent \\
\qquad\quad \wedge\, \{R_{IMP}(P.1, ic, oc), R_{IMP}(Q.1, oc, ic)\} \subseteq FOP \\
\qquad\quad \wedge\, R_{IMP}(P.1, ic, oc) \approx R_{IMP}(Q.1, oc, ic)\,) \\
\quad \bullet P\,[\,ics \leftrightarrow_+ ocs\,]\,Q = P\,[\,ics \asymp ocs\,]\,Q
\end{array}$$

I.3.6 Extended Feedback Composition

Definition I.19 (Extended feedback composition) *Let P be a component contract, and* $chseq_1$ *and* $chseq_2$ *two nonempty sequences of distinct communication channels, such that:*

- $(\mathrm{ran}\, chseq_1 \cup \mathrm{ran}\, chseq_2) \subseteq \mathcal{C}_P \wedge (\mathrm{ran}\, chseq_1 \cap \mathrm{ran}\, chseq_2) = \emptyset$*;*
- $(\mathrm{ran}\, chseq_1 \cup \mathrm{ran}\, chseq_2)$ *DecoupledIn* $P \wedge \# chseq_1 = \# chseq_2$*;*

- $\forall\, i, z, c \mid c = chseq_1(i) \wedge z = chseq_2(i) \bullet Prot_{IMP}(P, c) \, [\![R_{IO}^{c \to z}]\!]$ *and* $Prot_{IMP}(P, z)[\![R_{IO}^{z \to c}]\!]$ *are I/O confluent strong compatible port-protocols and satisfy the finite output property;*

Then, the extended feedback composition of P *(namely* $P[chseq_1 \hookrightarrow chseq_2]$*) via the channels within* $chseq_P$ *and* $chseq_Q$ *is defined as follows:*

$$P[chseq_1 \hookrightarrow chseq_2] = P \asymp|_{chseq_2}^{chseq_1}$$

Formally

function$(_[_ \hookrightarrow_+ _])$

$$\begin{array}{|l}
\_[\_ \hookrightarrow_+ \_] : CTR \times \mathrm{iseq}\, CHANNEL \times \mathrm{iseq}\, CHANNEL \nrightarrow CTR \\
\hline
\forall\, P : CTR;\ ics, ocs : \mathrm{seq}\, CHANNEL \\
\quad \mid (\mathrm{ran}\, ics \cup \mathrm{ran}\, ocs \subseteq \mathcal{C}(P) \\
\qquad \wedge\ \mathrm{ran}\, ics \cap \mathrm{ran}\, ocs = \emptyset \\
\qquad \wedge\ \mathrm{ran}\, ics \cup \mathrm{ran}\, ocs\ DecoupledIn\ P.1 \\
\qquad \wedge\ \#ics = \#ocs \\
\qquad \forall\, i : \mathrm{dom}\, ics;\ ic, oc : CHANNEL \\
\qquad\quad \mid ic = ics(i) \wedge oc = ocs(i) \\
\qquad\quad \bullet \{R_{IMP}(P.1, ic, oc), R_{IMP}(P.1, oc, ic)\} \subseteq IOConfluent \\
\qquad\quad \wedge\ \{R_{IMP}(P.1, ic, oc), R_{IMP}(P.1, oc, ic)\} \subseteq FOP \\
\qquad\quad \wedge\ R_{IMP}(P.1, ic, oc) \approx R_{IMP}(P.1, oc, ic)) \\
\quad \bullet P[ocs \hookrightarrow_+ ics] = P \asymp| (ocs, ics)
\end{array}$$

I.3.7 Wrapping interaction

Definition I.20 (Wrapping interaction) *Let Ctr be an interaction component contract, and CC a set of communication channels, such that* $CC \notin IntCh_{Ctr}$*. Then the wrapping interaction version of Ctr (denoted by* $Ctr \lceil\lceil CC$*) is given by:*

$$Ctr \lceil\lceil CC = Ctr \setminus CC$$

Formally

function 30 leftassoc$(_ \lceil\lceil_i _)$

$$\_ \lceil\lceil_i \_ : CTR \times \mathbb{P}\, CHANNEL \nrightarrow CTR$$

$$\forall\, c : CTR;\ CC : \mathbb{P}\, CHANNEL$$
$$\mid CC \subseteq IOChannels \land CC \cap IntCh(c) = \emptyset$$
$$\bullet\, c \lceil\lceil_i CC = c \lceil\lceil CC$$

I.4 Extending the Model with Metadata

I.4.1 Enriched component contract

Definition 3.6 (Enriched component contract) *Let Ctr be a protocol oriented component contract, and $\mathcal{K}$ a metadata derived from its elements. An enriched component contract that includes Ctr is represented by:*

$$\langle \mathcal{B}_{Ctr}, \mathcal{R}_{Ctr}, \mathcal{I}_{Ctr}, \mathcal{C}_{Ctr}, \mathcal{K} \rangle$$

where $\mathcal{K}$ comprises the following information:

$$\mathcal{K} : \langle Prot^{\mathcal{K}}, CTX^{\mathcal{K}}, DProt^{\mathcal{K}}, Dec^{\mathcal{K}} \rangle$$

such that:

- $\operatorname{dom} Prot^{\mathcal{K}} \subseteq \mathcal{C}_{Ctr} \wedge \forall c : \operatorname{dom} Prot^{\mathcal{K}} \bullet Prot^{\mathcal{K}}(c) \sqsubseteq_{\mathrm{F}} Prot_{IMP}(Ctr, c)$
- $\operatorname{dom} DProt^{\mathcal{K}} \subseteq \mathcal{C}_{Ctr} \wedge \forall c : \operatorname{dom} DProt^{\mathcal{K}} \bullet DProt^{\mathcal{K}}(c)$ is the dual protocol of $Prot^{\mathcal{K}}(c)$
- $\operatorname{dom} CTX^{\mathcal{K}} \subseteq \mathcal{C}_{Ctr} \wedge \forall c : \operatorname{dom} CTX^{\mathcal{K}} \bullet CTX^{\mathcal{K}}(c)$ is the context process of $Prot^{\mathcal{K}}(c)$
- $\operatorname{dom} Dec^{\mathcal{K}} \subseteq \mathcal{C}_{Ctr} \wedge \operatorname{ran} Dec^{\mathcal{K}} \subseteq \mathcal{C}_{Ctr} \wedge$
  $\forall c_1, c_2 : \mathcal{C}_{Ctr} \bullet c_1\ Dec^{\mathcal{K}}\ c_2 \Rightarrow \{c_1, c_2\}\ DecoupledIn\ Ctr \wedge c_2\ Dec^{\mathcal{K}}\ c_1$

Formally

$$\begin{array}{|l}
\mathcal{K} : \mathbb{P}((CHANNEL \pfun CIRCUS\_PROCESS) \\
\qquad\qquad \times (CHANNEL \pfun CIRCUS\_PROCESS) \\
\qquad\qquad \times (CHANNEL \pfun CIRCUS\_PROCESS) \\
\qquad\qquad \times (CHANNEL \leftrightarrow CHANNEL))
\end{array}$$

$$\begin{array}{|l}
Prot : \mathcal{K} \rightarrow (CHANNEL \pfun CIRCUS\_PROCESS) \\
\hline
\forall k : \mathcal{K} \bullet Prot(k) = k.1
\end{array}$$

$$\begin{array}{|l}
DProt : \mathcal{K} \rightarrow (CHANNEL \pfun CIRCUS\_PROCESS) \\
\hline
\forall k : \mathcal{K} \bullet DProt(k) = k.2
\end{array}$$

$$
\begin{array}{|l}
CTX : \mathcal{K} \rightarrow (CHANNEL \nrightarrow CIRCUS\_PROCESS) \\
\hline
\forall\, k : \mathcal{K} \bullet CTX(k) = k.3
\end{array}
$$

$$
\begin{array}{|l}
Dec : \mathcal{K} \rightarrow (CHANNEL \leftrightarrow CHANNEL) \\
\hline
\forall\, k : \mathcal{K} \bullet Dec(k) = k.4
\end{array}
$$

$$
\begin{array}{|l}
CTR_{+} : \mathbb{P}(CIRCUS\_PROCESS \times (CHANNEL \rightarrow TYPE) \times \\
\qquad\qquad (\mathbb{P}\, TYPE) \times (\mathbb{P}\, CHANNEL) \times \mathcal{K}) \\
\hline
\forall\, B : CIRCUS\_PROCESS;\ R : CHANNEL \rightarrow TYPE; \\
\quad I : \mathbb{P}\, TYPE;\ C : \mathbb{P}\, CHANNEL;\ K : \mathcal{K} \\
\quad \mid (B, R, I, C) \in CTR \\
\qquad \bullet (B, R, I, C, K) \in CTR_{+} \\
\qquad\quad \Leftrightarrow \\
\qquad\quad (\mathrm{dom}(Prot(K)) \subseteq C \land \\
\qquad\qquad \forall\, c : \mathrm{dom}(Prot(K)) \bullet (Prot(K))(c) \sqsubseteq_{\mathrm{F}} Prot_{IMP}(B, c)) \\
\qquad\quad \land (\mathrm{dom}(DProt(K)) \subseteq C \land \\
\qquad\qquad \forall\, c : \mathrm{dom}(DProt(K)) \bullet (DProt(K))(c) = DP((Prot(K))(c))) \\
\qquad\quad \land (\mathrm{dom}(CTX(K)) \subseteq C \land \\
\qquad\qquad \forall\, c : \mathrm{dom}(CTX(K)) \bullet (CTX(K))(c) = CTX((Prot(K))(c))) \\
\qquad\quad \land (\mathrm{dom}(Dec(K)) \subseteq C \land \mathrm{ran}(Dec(K)) \subseteq C \\
\qquad\qquad \forall\, c_1, c_2 : C \bullet (c_1, c_2) \in Dec(K) \Rightarrow \{c_1, c_2\}\ DecoupledIn\ B)
\end{array}
$$

$$
\begin{array}{|l}
\mathcal{B}_{+} : CTR_{+} \rightarrow CIRCUS\_PROCESS \\
\hline
\forall\, p_{+} : CTR_{+} \bullet \mathcal{B}_{+}(p_{+}) = p_{+}.1
\end{array}
$$

$$
\begin{array}{|l}
\mathcal{R}_{+} : CTR_{+} \rightarrow (CHANNEL \rightarrow TYPE) \\
\hline
\forall\, p_{+} : CTR_{+} \bullet \mathcal{R}_{+}(p_{+}) = p_{+}.2
\end{array}
$$

$$
\begin{array}{|l}
\mathcal{I}_{+} : CTR_{+} \rightarrow \mathbb{P}\, TYPE \\
\hline
\forall\, p_{+} : CTR_{+} \bullet \mathcal{I}_{+}(p_{+}) = p_{+}.3
\end{array}
$$

$$
\begin{array}{|l}
\mathcal{C}_{+} : CTR_{+} \rightarrow \mathbb{P}\, CHANNEL \\
\hline
\forall\, p_{+} : CTR_{+} \bullet \mathcal{C}_{+}(p_{+}) = p_{+}.4
\end{array}
$$

$$
\begin{array}{|l}
\mathcal{K}_{+} : CTR_{+} \rightarrow \mathcal{K} \\
\hline
\forall\, p_{+} : CTR_{+} \bullet \mathcal{K}_{+}(p_{+}) = p_{+}.5
\end{array}
$$

I.4.2 Enrich component contract

Definition I.21 (Enrich Component Contract) *Let Ctr be a protocol oriented component contract, and $\mathcal{K}$ a metadata derived from its elements. Then:*

$$Enrich(Ctr, \mathcal{K}) = \langle \mathcal{B}_{Ctr}, \mathcal{R}_{Ctr}, \mathcal{I}_{Ctr}, \mathcal{C}_{Ctr}, \mathcal{K} \rangle$$

$$\begin{array}{|l}
enrich : (CTR \times \mathcal{K}) \nrightarrow CTR_+ \\
\hline
\forall\, ctr : CTR;\ K : \mathcal{K} \\
\quad \mid (\mathcal{B}(ctr), \mathcal{R}(ctr), \mathcal{I}(ctr), \mathcal{C}(ctr), K) \in CTR_+ \\
\quad \bullet\ enrich\,(ctr, K) = (\mathcal{B}(ctr), \mathcal{R}(ctr), \mathcal{I}(ctr), \mathcal{C}(ctr), K)
\end{array}$$

I.4.3 Enriched interleaving composition

Definition 3.7 (Enriched interleaving composition) *Let P and Q be two enriched component contracts, such that P and Q have disjoint channels, $\mathcal{C}_P \cap \mathcal{C}_Q = \emptyset$. Then, the enriched interleaving composition of P and Q (namely $P\,[|||]\,Q$) is given by:*

$$\begin{aligned} P\,[|||]\,Q = Enrich(&\langle \mathcal{B}_P, \mathcal{R}_P, \mathcal{I}_P, \mathcal{C}_P \rangle_{\langle\rangle} \asymp {}_{\langle\rangle}\langle \mathcal{B}_Q, \mathcal{R}_Q, \mathcal{I}_Q, \mathcal{C}_Q \rangle, \\ &\langle Prot^{\mathcal{K}}_{PQ}, CTX^{\mathcal{K}}_{PQ}, DProt^{\mathcal{K}}_{PQ}, Dec^{\mathcal{K}}_{PQ} \rangle) \end{aligned}$$

where

(i) $Prot^{\mathcal{K}}_{PQ} = Prot^{\mathcal{K}}_{P} \cup Prot^{\mathcal{K}}_{Q}$

(ii) $CTX^{\mathcal{K}}_{PQ} = CTX^{\mathcal{K}}_{P} \cup CTX^{\mathcal{K}}_{Q}(c)$

(iii) $DProt^{\mathcal{K}}_{PQ} = DProt^{\mathcal{K}}_{P} \cup DProt^{\mathcal{K}}_{Q}$

(iv) $Dec^{\mathcal{K}}_{PQ} = Dec^{\mathcal{K}}_{P} \cup Dec^{\mathcal{K}}_{Q} \cup \{(c_1, c_2) \mid (c_1 \in \mathcal{C}_Q \wedge c_2 \in \mathcal{C}_P) \vee (c_1 \in \mathcal{C}_P \wedge c_2 \in \mathcal{C}_Q)\}$

Formally

function 30 leftassoc($\_\,[|||]_+\_$)

$$
\begin{array}{|l}
\_\,[|||]_{+}\,\_ : (CTR_{+} \times CTR_{+}) \nrightarrow CTR_{+} \\
\hline
\forall\, P, Q : CTR_{+} \\
\quad \mid \mathcal{C}_{+}(P) \cap \mathcal{C}_{+}(Q) = \emptyset \\
\quad \bullet\; \exists\, Prot_{PQ}, DProt_{PQ}, CTX_{PQ} : CHANNEL \nrightarrow CIRCUS\_PROCESS; \\
\qquad Dec_{PQ} : CHANNEL \leftrightarrow CHANNEL \\
\qquad \mid Prot_{PQ} = Prot(\mathcal{K}_{+}(P)) \cup Prot(\mathcal{K}_{+}(Q)) \\
\qquad \wedge\; DProt_{PQ} = DProt(\mathcal{K}_{+}(P)) \cup DProt(\mathcal{K}_{+}(Q)) \\
\qquad \wedge\; CTX_{PQ} = CTX(\mathcal{K}_{+}(P)) \cup CTX(\mathcal{K}_{+}(Q)) \\
\qquad \wedge\; Dec_{PQ} = Dec(\mathcal{K}_{+}(P)) \cup Dec(\mathcal{K}_{+}(Q)) \cup \\
\qquad\qquad \{c_1, c_2 : CHANNEL \mid (c_1 \in \mathcal{C}_{+}(P) \wedge c_2 \in \mathcal{C}_{+}(Q)) \\
\qquad\qquad\qquad \vee\, (c_1 \in \mathcal{C}_{+}(Q) \wedge c_2 \in \mathcal{C}_{+}(P))\} \\
\qquad \bullet\; P\,[|||]_{+}\,Q = \\
\qquad\quad enrich((\mathcal{B}_{+}(P), \mathcal{R}_{+}(P), \mathcal{I}_{+}(P), \mathcal{C}_{+}(P)) \\
\qquad\qquad [\,\langle\rangle \asymp \langle\rangle\,] \\
\qquad\qquad (\mathcal{B}_{+}(Q), \mathcal{R}_{+}(Q), \mathcal{I}_{+}(Q), \mathcal{C}_{+}(Q)), \\
\qquad\qquad (Prot_{PQ}, DProt_{PQ}, CTX_{PQ}, Dec(\mathcal{K}_{+}(P))))
\end{array}
$$

I.4.4 Enriched Communication Composition

Definition 3.8 (Enriched Communication composition) *Let P and Q be two enriched component contracts, and ic and oc two communication channels, such that $ic \in \mathcal{C}_P \wedge oc \in \mathcal{C}_Q$, $\mathcal{C}_P \cap \mathcal{C}_Q = \emptyset$, and the port-protocols $Prot^{\mathcal{K}}_{P}(ic) \llbracket R^{ic\rightarrow oc}_{IO} \rrbracket$ and $Prot^{\mathcal{K}}_{Q}(oc) \llbracket R^{oc\rightarrow ic}_{IO} \rrbracket$ are I/O confluent strong compatible and satisfy the finite output property. Then, the communication composition of P and Q (namely $P[ic \leftrightarrow oc]Q$) via ic and oc is defined as follows:*

$$
\begin{aligned}
P[ic \leftrightarrow oc]Q = Enrich(&\langle \mathcal{B}_P, \mathcal{R}_P, \mathcal{I}_P, \mathcal{C}_P\rangle_{\langle ic\rangle} \asymp_{\langle oc\rangle} \langle \mathcal{B}_Q, \mathcal{R}_Q, \mathcal{I}_Q, \mathcal{C}_Q\rangle, \\
&\langle Prot^{\mathcal{K}}_{PQ}, CTX^{\mathcal{K}}_{PQ}, DProt^{\mathcal{K}}_{PQ}, Dec^{\mathcal{K}}_{PQ}\rangle)
\end{aligned}
$$

where

$$
\begin{aligned}
Prot^{\mathcal{K}}_{PQ} &= \{c \mapsto Prot^{\mathcal{K}}_{P}(c) \mid c \in \operatorname{dom}(Prot^{\mathcal{K}}_{P}(c)) \setminus \{ic\}\} \\
&\quad \cup \{c \mapsto Prot^{\mathcal{K}}_{Q}(c) \mid c \in \operatorname{dom}(Prot^{\mathcal{K}}_{Q}(c)) \setminus \{oc\}\} \\
DProt^{\mathcal{K}}_{PQ} &= \{c \mapsto DProt^{\mathcal{K}}_{P}(c) \mid c \in \operatorname{dom}(DProt^{\mathcal{K}}_{P}(c)) \setminus \{ic\}\} \\
&\quad \cup \{c \mapsto DProt^{\mathcal{K}}_{Q}(c) \mid c \in \operatorname{dom}(DProt^{\mathcal{K}}_{Q}(c)) \setminus \{oc\}\} \\
CTX^{\mathcal{K}}_{PQ} &= \{c \mapsto CTX^{\mathcal{K}}_{P}(c) \mid c \in \operatorname{dom}(CTX^{\mathcal{K}}_{P}(c)) \setminus \{ic\}\} \\
&\quad \cup \{c \mapsto CTX^{\mathcal{K}}_{Q}(c) \mid c \in \operatorname{dom}(CTX^{\mathcal{K}}_{Q}(c)) \setminus \{oc\}\} \\
Dec^{\mathcal{K}}_{PQ} &= \{(c_1, c_2) \mid \{c_1, c_2\} \cap \{ic, oc\} = \emptyset \\
&\qquad \wedge (((c_1\, Dec^{\mathcal{K}}_{P}\, ic \vee ic\, Dec^{\mathcal{K}}_{P}\, c_1) \wedge (c_2 \in \mathcal{C}_Q \vee c_1\, Dec^{\mathcal{K}}_{P}\, c_2)) \\
&\qquad\quad \vee ((oc\, Dec^{\mathcal{K}}_{Q}\, c_2 \vee c_2\, Dec^{\mathcal{K}}_{Q}\, oc) \wedge (c_1 \in \mathcal{C}_P \vee c_1\, Dec^{\mathcal{K}}_{Q}\, c_2)))\}
\end{aligned}
$$

Formally

function 30 leftassoc(_ [_ ↔₊ _] _)

$$
\begin{array}{|l}
\_\,[\_ \leftrightarrow_{+} \_\,]\,\_ : CTR_{+} \times CHANNEL \times CHANNEL \times CTR_{+} \pfun CTR_{+} \\
\hline
\forall P, Q : CTR_{+};\ ic, oc : CHANNEL \\
\quad \mid (\, ic \in \mathcal{C}_{+}(P) \land oc \in \mathcal{C}_{+}(Q) \land \mathcal{C}_{+}(P) \cap \mathcal{C}_{+}(Q) = \emptyset \\
\qquad \land \{R_{IMP}(\mathcal{B}_{+}(P), ic, oc), R_{IMP}(\mathcal{B}_{+}(Q), oc, ic)\} \subseteq IOConfluent \\
\qquad \land \{R_{IMP}(\mathcal{B}_{+}(P), ic, oc), R_{IMP}(\mathcal{B}_{+}(Q), oc, ic)\} \subseteq FOP \\
\qquad \land R_{IMP}(\mathcal{B}_{+}(P), ic, oc) \approx R_{IMP}(\mathcal{B}_{+}(Q), oc, ic)\,) \\
\quad \bullet \exists\, Prot_{PQ}, DProt_{PQ}, CTX_{PQ} : CHANNEL \pfun CIRCUS\_PROCESS; \\
\qquad Dec_{PQ} : CHANNEL \leftrightarrow CHANNEL \\
\qquad \mid Prot_{PQ} = \{c : \mathrm{dom}(Prot(\mathcal{K}_{+}(P))) \setminus \{ic\} \bullet c \mapsto (Prot(\mathcal{K}_{+}(P))(c))\} \\
\qquad\qquad\qquad \cup \{c : \mathrm{dom}(Prot(\mathcal{K}_{+}(Q))) \setminus \{oc\} \bullet c \mapsto (Prot(\mathcal{K}_{+}(Q))(c))\} \\
\qquad \land DProt_{PQ} = \{c : \mathrm{dom}(DProt(\mathcal{K}_{+}(P))) \setminus \{ic\} \bullet c \mapsto (DProt(\mathcal{K}_{+}(P))(c))\} \\
\qquad\qquad\qquad \cup \{c : \mathrm{dom}(DProt(\mathcal{K}_{+}(Q))) \setminus \{oc\} \bullet c \mapsto (DProt(\mathcal{K}_{+}(Q))(c))\} \\
\qquad \land CTX_{PQ} = \{c : \mathrm{dom}(CTX(\mathcal{K}_{+}(P))) \setminus \{ic\} \bullet c \mapsto (CTX(\mathcal{K}_{+}(P))(c))\} \\
\qquad\qquad\qquad \cup \{c : \mathrm{dom}(CTX(\mathcal{K}_{+}(Q))) \setminus \{oc\} \bullet c \mapsto (CTX(\mathcal{K}_{+}(Q))(c))\} \\
\qquad \land Dec_{PQ} = \{\, c_1, c_2 : CHANNEL \mid \\
\qquad\qquad\qquad \{c_1, c_2\} \cap \{ic, oc\} = \emptyset \\
\qquad\qquad\qquad \land (\,(((c_1, ic) \in Dec(\mathcal{K}_{+}(P)) \lor (ic, c_1) \in Dec(\mathcal{K}_{+}(P))) \\
\qquad\qquad\qquad\qquad \land (c_2 \in \mathcal{C}_{+}(Q) \lor (c_1, c_2) \in Dec(\mathcal{K}_{+}(P)))) \\
\qquad\qquad\qquad\quad \lor (((oc, c_2) \in Dec(\mathcal{K}_{+}(Q)) \lor (c_2, oc) \in Dec(\mathcal{K}_{+}(Q))) \\
\qquad\qquad\qquad\qquad \land (c_1 \in \mathcal{C}_{+}(P) \lor (c_1, c_2) \in Dec(\mathcal{K}_{+}(Q)))))\,)\,\} \\
\quad \bullet P\,[\,ic \leftrightarrow_{+} oc\,]\,Q = \\
\qquad\qquad enrich((\mathcal{B}_{+}(P), \mathcal{R}_{+}(P), \mathcal{I}_{+}(P), \mathcal{C}_{+}(P)) \\
\qquad\qquad\qquad [\,\langle ic \rangle \asymp \langle oc \rangle\,] \\
\qquad\qquad\qquad (\mathcal{B}_{+}(Q), \mathcal{R}_{+}(Q), \mathcal{I}_{+}(Q), \mathcal{C}_{+}(Q)), \\
\qquad\qquad\qquad (Prot_{PQ}, DProt_{PQ}, CTX_{PQ}, Dec(\mathcal{K}_{+}(P))))
\end{array}
$$

I.4.5 Enriched Feedback Composition

Definition I.22 (Enriched Feedback composition) *Let P be an enriched component contract, and ic and oc two communication channels, such that $\{ic, oc\} \subseteq \mathcal{C}_P$, and the port-protocols $Prot^{\mathcal{K}}_P(ic) \parallel [\, R^{ic \to oc}_{IO}]\!\!\parallel$ and $Prot^{\mathcal{K}}_P(oc) \parallel [\, R^{oc \to ic}_{IO}]\!\!\parallel$ are I/O confluent strong compatible and satisfy the finite output property, and $ic\ Dec^{\mathcal{K}}_P\ oc$. Then, the feedback composition P (namely $P[oc \hookrightarrow ic]$) hooking oc to ic is defined as follows:*

$$P[oc \hookrightarrow ic] = Enrich(\langle \mathcal{B}_P, \mathcal{R}_P, \mathcal{I}_P, \mathcal{C}_P \rangle \asymp|^{\langle ic \rangle}_{\langle oc \rangle}, \\ \langle Prot^{\mathcal{K}}_S, CTX^{\mathcal{K}}_S, DProt^{\mathcal{K}}_S, Dec^{\mathcal{K}}_S \rangle)$$

where

$$\begin{aligned}
&\mathit{Prot}_S^{\mathcal{K}} = \{c \mapsto \mathit{Prot}_P^{\mathcal{K}}(c) \mid c \in \mathrm{dom}(\mathit{Prot}_P^{\mathcal{K}}(c)) \setminus \{ic, oc\}\}\\
&\mathit{DProt}_S^{\mathcal{K}} = \{c \mapsto \mathit{DProt}_P^{\mathcal{K}}(c) \mid c \in \mathrm{dom}(\mathit{DProt}_P^{\mathcal{K}}(c)) \setminus \{ic, oc\}\}\\
&\mathit{CTX}_S^{\mathcal{K}} = \{c \mapsto \mathit{CTX}_P^{\mathcal{K}}(c) \mid c \in \mathrm{dom}(\mathit{CTX}_P^{\mathcal{K}}(c)) \setminus \{ic, oc\}\}\\
&\mathit{Dec}_S^{\mathcal{K}} = \{(c_1, c_2) \mid \{c_1, c_2\} \cap \{ic, oc\} = \emptyset \wedge c_1\, \mathit{Dec}_P^{\mathcal{K}}\, c_2\\
&\qquad\qquad \wedge (((c_1\, \mathit{Dec}_P^{\mathcal{K}}\, ic \wedge c_1\, \mathit{Dec}_P^{\mathcal{K}}\, oc))\\
&\qquad\qquad\qquad \vee ((ic\, \mathit{Dec}_P^{\mathcal{K}}\, c_2 \wedge oc\, \mathit{Dec}_P^{\mathcal{K}}\, c_2)))\}
\end{aligned}$$

Formally

function(_ [_ $\hookrightarrow_+$ _])

$$\begin{aligned}
&\_[\_ \hookrightarrow_+ \_] : \mathit{CTR}_+ \times \mathit{CHANNEL} \times \mathit{CHANNEL} \rightarrow\!\!\!\!\!+\; \mathit{CTR}_+\\
\hline
&\forall P : \mathit{CTR}_+;\ ic, oc : \mathit{CHANNEL}\\
&\quad \mid (\{ic, oc\} \subseteq \mathcal{C}_+(P)\\
&\qquad \wedge \{R_{\mathit{IMP}}(\mathcal{B}_+(P), ic, oc), R_{\mathit{IMP}}(\mathcal{B}_+(P), oc, ic)\} \subseteq \mathit{IOConfluent}\\
&\qquad \wedge \{R_{\mathit{IMP}}(\mathcal{B}_+(P), ic, oc), R_{\mathit{IMP}}(\mathcal{B}_+(P), oc, ic)\} \subseteq \mathit{FOP}\\
&\qquad \wedge R_{\mathit{IMP}}(\mathcal{B}_+(P), ic, oc) \approx R_{\mathit{IMP}}(\mathcal{B}_+(P), oc, ic)\\
&\qquad \wedge (ic, oc) \in \mathit{Dec}(\mathcal{K}_+(P)))\\
&\quad \bullet \exists\, \mathit{Prot}_S, \mathit{DProt}_S, \mathit{CTX}_S : \mathit{CHANNEL} \rightarrow\!\!\!\!\!+\; \mathit{CIRCUS\_PROCESS};\\
&\qquad \mathit{Dec}_S : \mathit{CHANNEL} \leftrightarrow \mathit{CHANNEL}\\
&\qquad \mid \mathit{Prot}_S = \{c : \mathrm{dom}(\mathit{Prot}(\mathcal{K}_+(P))) \setminus \{ic, oc\} \bullet c \mapsto (\mathit{Prot}(\mathcal{K}_+(P))(c))\}\\
&\qquad \wedge \mathit{DProt}_S = \{c : \mathrm{dom}(\mathit{DProt}(\mathcal{K}_+(P))) \setminus \{ic, oc\} \bullet c \mapsto (\mathit{DProt}(\mathcal{K}_+(P))(c))\}\\
&\qquad \wedge \mathit{CTX}_S = \{c : \mathrm{dom}(\mathit{CTX}(\mathcal{K}_+(P))) \setminus \{ic, oc\} \bullet c \mapsto (\mathit{CTX}(\mathcal{K}_+(P))(c))\}\\
&\qquad \wedge \mathit{Dec}_S = \{\, c_1, c_2 : \mathit{CHANNEL} \mid\\
&\qquad\qquad\qquad \{c_1, c_2\} \cap \{ic, oc\} = \emptyset\\
&\qquad\qquad\qquad \wedge (ic, oc) \in \mathit{Dec}(\mathcal{K}_+(P))\\
&\qquad\qquad\qquad \wedge (\,(((c_1, ic) \in \mathit{Dec}(\mathcal{K}_+(P)) \wedge (c_1, oc) \in \mathit{Dec}(\mathcal{K}_+(P))))\\
&\qquad\qquad\qquad\quad \vee (((ic, c_2) \in \mathit{Dec}(\mathcal{K}_+(P)) \wedge (oc, c_2) \in \mathit{Dec}(\mathcal{K}_+(P))))\,)\,\}\\
&\quad \bullet P\,[\, oc \hookrightarrow_+ ic\,] =\\
&\qquad\qquad \mathit{enrich}((\mathcal{B}_+(P), \mathcal{R}_+(P), \mathcal{I}_+(P), \mathcal{C}_+(P)) \asymp\!| (\langle oc \rangle, \langle ic \rangle),\\
&\qquad\qquad\qquad (\mathit{Prot}_S, \mathit{DProt}_S, \mathit{CTX}_S, \mathit{Dec}(\mathcal{K}_+(P))))
\end{aligned}$$

I.4.6 Enriched Reflexive Composition

Definition I.23 (Enriched Reflexive composition) *Let P be an enriched component contract, and ic and oc two communication channels, such that*

$\{ic, oc\} \subseteq \mathcal{C}_P$, and $P \restriction \{c, z\}$ buffering self-injection compatible and satisfies the finite output property. Then, the reflexive composition P (namely $P[oc \hookrightarrow ic]$) hooking oc to ic is defined as follows:

$$\begin{array}{l} P[oc \hookrightarrow ic] = \mathit{Enrich}(\langle \mathcal{B}_P, \mathcal{R}_P, \mathcal{I}_P, \mathcal{C}_P \rangle \asymp\!\!|^{\langle ic \rangle}_{\langle oc \rangle}, \\ \qquad\qquad\qquad\qquad \langle \mathit{Prot}^{\mathcal{K}}_S, \mathit{CTX}^{\mathcal{K}}_S, \mathit{DProt}^{\mathcal{K}}_S, \mathit{Dec}^{\mathcal{K}}_S \rangle) \end{array}$$

where

$$\begin{array}{l} \mathit{Prot}^{\mathcal{K}}_S = \{c \mapsto \mathit{Prot}^{\mathcal{K}}_P(c) \mid c \in \mathrm{dom}(\mathit{Prot}^{\mathcal{K}}_P(c)) \setminus \{ic, oc\}\} \\ \mathit{DProt}^{\mathcal{K}}_S = \{c \mapsto \mathit{DProt}^{\mathcal{K}}_P(c) \mid c \in \mathrm{dom}(\mathit{DProt}^{\mathcal{K}}_P(c)) \setminus \{ic, oc\}\} \\ \mathit{CTX}^{\mathcal{K}}_S = \{c \mapsto \mathit{CTX}^{\mathcal{K}}_P(c) \mid c \in \mathrm{dom}(\mathit{CTX}^{\mathcal{K}}_P(c)) \setminus \{ic, oc\}\} \\ \mathit{Dec}^{\mathcal{K}}_S = \{(c_1, c_2) \mid \{c_1, c_2\} \cap \{ic, oc\} = \emptyset \wedge c_1\, \mathit{Dec}^{\mathcal{K}}_P\, c_2 \\ \qquad\qquad\qquad\qquad \wedge (((c_1\, \mathit{Dec}^{\mathcal{K}}_P\, ic \wedge c_1\, \mathit{Dec}^{\mathcal{K}}_P\, oc)) \\ \qquad\qquad\qquad\qquad\qquad \vee ((ic\, \mathit{Dec}^{\mathcal{K}}_P\, c_2 \wedge oc\, \mathit{Dec}^{\mathcal{K}}_P\, c_2)))\} \end{array}$$

Formally

function(_ [_ $\hookrightarrow_+$ _])

$$\begin{array}{|l} \_ [\_ \hookrightarrow_+ \_] : \mathit{CTR}_+ \times \mathit{CHANNEL} \times \mathit{CHANNEL} \nrightarrow \mathit{CTR}_+ \\ \hline \forall P : \mathit{CTR}_+;\ ic, oc : \mathit{CHANNEL} \\ \quad \mid (\{ic, oc\} \subseteq \mathcal{C}_+(P) \\ \qquad \wedge \mathcal{B}_+(P)\ \mathit{buffSelfInjComp}\,(ic, oc) \\ \qquad \wedge \mathcal{B}_+(P) \restriction_B \{ic, oc\} \in \mathit{FOP}) \\ \quad \bullet \exists \mathit{Prot}_S, \mathit{DProt}_S, \mathit{CTX}_S : \mathit{CHANNEL} \nrightarrow \mathit{CIRCUS\_PROCESS}; \\ \qquad \mathit{Dec}_S : \mathit{CHANNEL} \leftrightarrow \mathit{CHANNEL} \\ \qquad \mid \mathit{Prot}_S = \{c : \mathrm{dom}(\mathit{Prot}(\mathcal{K}_+(P))) \setminus \{ic, oc\} \bullet c \mapsto (\mathit{Prot}(\mathcal{K}_+(P))(c))\} \\ \qquad \wedge \mathit{DProt}_S = \{c : \mathrm{dom}(\mathit{DProt}(\mathcal{K}_+(P))) \setminus \{ic, oc\} \bullet c \mapsto (\mathit{DProt}(\mathcal{K}_+(P))(c))\} \\ \qquad \wedge \mathit{CTX}_S = \{c : \mathrm{dom}(\mathit{CTX}(\mathcal{K}_+(P))) \setminus \{ic, oc\} \bullet c \mapsto (\mathit{CTX}(\mathcal{K}_+(P))(c))\} \\ \qquad \wedge \mathit{Dec}_S = \{\ c_1, c_2 : \mathit{CHANNEL} \mid \\ \qquad\qquad\qquad\qquad \{c_1, c_2\} \cap \{ic, oc\} = \emptyset \\ \qquad\qquad\qquad\qquad \wedge (ic, oc) \in \mathit{Dec}(\mathcal{K}_+(P)) \\ \qquad\qquad\qquad\qquad \wedge (\ (((c_1, ic) \in \mathit{Dec}(\mathcal{K}_+(P)) \wedge (c_1, oc) \in \mathit{Dec}(\mathcal{K}_+(P)))) \\ \qquad\qquad\qquad\qquad\qquad \vee (((ic, c_2) \in \mathit{Dec}(\mathcal{K}_+(P)) \wedge (oc, c_2) \in \mathit{Dec}(\mathcal{K}_+(P))))\ )\ \} \\ \quad \bullet P\,[\,oc \hookrightarrow_+ ic\,] = \\ \qquad\qquad\qquad \mathit{enrich}((\mathcal{B}_+(P), \mathcal{R}_+(P), \mathcal{I}_+(P), \mathcal{C}_+(P))\ \asymp\!\!|\ (\langle oc \rangle, \langle ic \rangle), \\ \qquad\qquad\qquad\qquad (\mathit{Prot}_S, \mathit{DProt}_S, \mathit{CTX}_S, \mathit{Dec}(\mathcal{K}_+(P)))) \end{array}$$

J Proofs on Model Equivalence

In this section, we demonstrate the correctness of the mapping function Υ that translates ***Circus*** into CSP processes. We consider *Skip*, *Stop*, *Chaos*, prefixing, external and internal choice, guarded actions, sequence, parallel composition and interleaving.

We make use of the following:

- The definition of *traces* and *failures* are those from [Ros98]
- The definition of Σ is from [Ros98]: the set containing all events but ✓
- The definition of $\Sigma^{\checkmark}$ is from [Ros98]: the set containing all events and ✓
- The definition of C is that from [Oli06]
- We adopt the notation from [Oli06]: A^b_c denotes $A[b/okay'][c/wait]$.
- In the UTP theory, ✓ is not allowed as an event.

The UTP observational variables are defined as follows:

- $tr, tr' : \operatorname{seq} \Sigma$
- $ref, ref' : \mathbb{P}\,\Sigma$
- $wait, wait', okay, okay' : \mathbb{B}$

Abbreviations in Proofs:

- PC: Predicate Calculus
- ST: Set Theory
- SC: Set Comprehension
- SS: Sequence Substitution
- IH: Inductive Hypothesis

Definitions from [Ros98]:

$$\begin{array}{l} failures(\mathtt{c} \rightarrow \mathtt{SKIP}) \mathrel{\widehat=} \\ \quad \{(\langle\rangle, X) \mid c \notin X \underline{\wedge\ X \subseteq \Sigma^{\checkmark}}\} \cup \{(\langle c \rangle \frown s, X) \mid (s, X) \in failures(\mathtt{SKIP})\} \end{array}$$

J.1 Lemmas from Oliveira's Phd

These lemmas are proved in [Oli06].

Lemma J.1 $(P \land g')\,;\, Q = P\,;\,(g \land Q)$ *provided g is a UTP condition.*

Lemma J.2

$$c?x : P \to A(x) \mathrel{\widehat=} \Box\, x : \{v : \delta(c) \mid P\} \bullet c.x \to A(x)$$

provided $\{v : \delta(c) \mid P\}$ is finite.

Lemma J.3 $(c \to Skip)_f = \boldsymbol{CSP1}(okay' \land do_{\mathcal{C}}(c, Sync) \land v' = v)$

Lemma J.4 $okay \land \boldsymbol{CSP1}(P) = okay \land P.$

Lemma J.5 $okay \land (\boldsymbol{CSP1}(P)\,;\, Q) = okay \land (P\,;\, Q)$

Lemma J.6 $(c \to A)_f^t = \boldsymbol{CSP1}(okay' \land do_{\mathcal{C}}(c, Sync) \land v' = v)\,;\, A^t$

Lemma J.7

$$\begin{array}{l} R(P_1 \vdash Q_1)\,;\, R(P_2 \vdash Q_2) \\ = \\ R(P_1 \land \neg\,((okay' \land \neg\, wait' \land Q_1)\,;\, \neg\, P_2) \vdash ((wait' \land Q_1) \lor ((okay' \land \neg\, wait' \land Q_1)\,;\, Q_2))) \end{array}$$

provided

- *P_1 does not mention any dashed variable*
- *P_1, Q_1, P_2 and Q_2 are R2*

Lemma J.8 $(\boldsymbol{R}(P \vdash Q))_f^t = \boldsymbol{CSP1}(\boldsymbol{R1}(\boldsymbol{R2}(P \Rightarrow Q)))$

Lemma J.9 $(\boldsymbol{R}(P \vdash Q))_f^f = \boldsymbol{R1}(\neg\, okay \land \boldsymbol{R2}(P))$

Lemma J.10 $p \land R2(P) = R2(p \land P)$ *provided p does not mention tr and tr'*

Lemma J.11

$$\begin{array}{rl} Skip^t = & (\neg\, okay \land tr \leq tr') \\ & \lor\, (tr' = tr \land wait' = wait \land v' = v \land (ref' = ref \lor \neg\, wait)) \end{array}$$

Lemma J.12 $Skip_f^t = \boldsymbol{CSP1}(tr' = tr \land \neg\, wait' \land v' = v)$

Lemma J.13 $Stop_f^t = \boldsymbol{CSP1}(tr' = tr \land wait')$

Lemma J.14

$$
\begin{array}{l}
(R(\exists\, s \bullet P[s, cs \cup ref'/tr', ref'] \land tr' - tr = s - tr \upharpoonright EVENT - cs))_f \\
= \exists\, s \bullet P[\langle\rangle, s, cs \cup ref'/tr, tr', ref'] \\
\qquad\qquad \land\ tr' - tr = s \upharpoonright EVENT - cs \\
\qquad\qquad \land\ tr \leq tr'
\end{array}
$$

J.2 Laws from UTP Tutorial Phd

These Laws are proved in [CW04].

UTP Law J.1 $\boldsymbol{R1}(P;\ Q) = P;\ Q$ *provided P and Q are* $\boldsymbol{R1}$*-healthy.*

UTP Law J.2 $\boldsymbol{R1}(P \lor Q) = P \lor Q$ *provided P and Q are* $\boldsymbol{R1}$*-healthy.*

UTP Law J.3 $\boldsymbol{R1}(P) \land Q = \boldsymbol{R1}(P \land Q)$

UTP Law J.4 $\boldsymbol{R2}(P;\ Q) = P;\ Q$ *provided P and Q are* $\boldsymbol{R2}$*-healthy.*

UTP Law J.5 $\boldsymbol{R2}(P \lor Q) = P \lor Q$ *provided P and Q are* $\boldsymbol{R2}$*-healthy.*

J.3 New Lemmas

Lemma J.15 *For every* ***Circus*** *action A such that $A = \boldsymbol{R}(P \vdash Q)$:*

$$\langle\rangle \in \{tr' - tr \mid okay \land Q\}$$

Proof. By induction on the syntax of ***Circus*** and the semantic functions. Informally, the semantics of all ***Circus*** actions are given as reactive designs whose post condition has at least one disjunct that keeps the traces unchanged $tr' = tr$.

Lemma J.16 *For every* ***Circus*** *action A.*

$$
\begin{array}{l}
okay \land ((okay' \land tr' = tr \frown s \land \neg\, wait' \land v' = v);\ A) \\
= \\
okay \land A_f[tr \frown s/tr]
\end{array}
$$

Proof.

$$
\begin{array}{ll}
okay \land ((okay' \land tr' = tr \frown s \land \neg\, wait' \land v' = v);\ A) & \text{[Lemma J.1]} \\
= okay \land ((tr' = tr \frown s \land v' = v);\ (okay \land \neg\, wait \land A)) & \text{[Definition of ;]}
\end{array}
$$

$$
\begin{array}{ll}
= okay \land & [\text{PC}] \\
\quad \exists\, okay_0, tr_0, wait_0, ref_0, v_0 \bullet & \\
\qquad tr_0 = tr \frown s \land v_0 = v & \\
\qquad \land\, okay_0 \land \neg\, wait_0 \land A[okay_0, tr_0, wait_0, ref_0, v_0 / okay, tr, wait, ref, v] & \\
= okay \land \exists\, ref_0 \bullet A[true, tr \frown s, false, ref_0 / okay, tr, wait, ref] & [\text{PC}] \\
= okay \land A[true, tr \frown s, false / okay, tr, wait] & [\text{PC}] \\
= okay \land A_f[tr \frown s / tr] &
\end{array}
$$

Lemma J.17

$$
\begin{array}{l}
(A \,\Box\, B)^t_f \\
= \\
\boldsymbol{CSP1}\left(\begin{array}{l}(\neg\, {A_1}^f_f \land \neg\, {A_2}^f_f) \\ \Rightarrow \\ (({A_1}^t_f \land {A_2}^t_f) \lhd tr' = tr \land wait' \rhd ({A_1}^t_f \lor {A_2}^t_f))\end{array}\right)
\end{array}
$$

provided A_1 and A_2 are ***R1****,* ***R2****.*

$$
\begin{array}{ll}
(A \,\Box\, B)^t_f & [\text{Choice}] \\
= \left(\mathbf{R}\left(\begin{array}{l}(\neg\, {A_1}^f_f \land \neg\, {A_2}^f_f) \\ \vdash \\ (({A_1}^t_f \land {A_2}^t_f) \lhd tr' = tr \land wait' \rhd ({A_1}^t_f \lor {A_2}^t_f))\end{array}\right)\right)^t_f & [\text{Lemma J.8}] \\
= \mathbf{CSP1}\left(\mathbf{R1}\left(\mathbf{R2}\left(\begin{array}{l}(\neg\, {A_1}^f_f \land \neg\, {A_2}^f_f) \\ \Rightarrow \\ (({A_1}^t_f \land {A_2}^t_f) \lhd tr' = tr \land wait' \rhd ({A_1}^t_f \lor {A_2}^t_f))\end{array}\right)\right)\right) & \\
& [\text{Tutorial - Laws 49 and 50 (proviso)}] \\
= \mathbf{CSP1}\left(\mathbf{R1}\left(\begin{array}{l}(\neg\, {A_1}^f_f \land \neg\, {A_2}^f_f) \\ \Rightarrow \\ (({A_1}^t_f \land {A_2}^t_f) \lhd tr' = tr \land wait' \rhd ({A_1}^t_f \lor {A_2}^t_f))\end{array}\right)\right) & \\
& [\text{Predicate calculus}] \\
= \mathbf{CSP1}\left(\mathbf{R1}\left(\begin{array}{l}{A_1}^f_f \\ \lor\, {A_2}^f_f \\ \lor\, (tr' = tr \land wait' \land {A_1}^t_f \land {A_2}^t_f) \\ \lor\, (\neg\, tr' = tr \land {A_1}^t_f) \\ \lor\, (\neg\, tr' = tr \land {A_2}^t_f) \\ \lor\, (\neg\, wait' \land {A_1}^t_f) \\ \lor\, (\neg\, wait' \land {A_2}^t_f)\end{array}\right)\right) &
\end{array}
$$

[Tutorial - Laws 37 and 38 (proviso)]

$$= \mathbf{CSP1}\left(\begin{array}{l} {A_1}_f^f \\ \lor {A_2}_f^f \\ \lor (tr' = tr \land wait' \land {A_1}_f^t \land {A_2}_f^t) \\ \lor (\neg\, tr' = tr \land {A_1}_f^t) \\ \lor (\neg\, tr' = tr \land {A_2}_f^t) \\ \lor (\neg\, wait' \land {A_1}_f^t) \\ \lor (\neg\, wait' \land {A_2}_f^t) \end{array}\right)$$

[Predicate calculus]

$$= \mathbf{CSP1}\left(\begin{array}{l} (\neg\, {A_1}_f^f \land \neg\, {A_2}_f^f) \\ \Rightarrow \\ (({A_1}_f^t \land {A_2}_f^t) \lhd tr' = tr \land wait' \rhd ({A_1}_f^t \lor {A_2}_f^t)) \end{array}\right)$$

Lemma J.18 $(R(P \vdash Q))^n = ok \land \boldsymbol{CSP1}(\boldsymbol{R1}(\boldsymbol{R2}(P \Rightarrow Q)))$

Proof.

$$\begin{array}{lr} (\mathbf{R}(P \vdash Q))^n & [A^n] \\ = ok \land \neg\, wt \land ok' \land \mathbf{R}(P \vdash Q) & [\text{PC}] \\ = ok \land (\mathbf{R}(P \vdash Q))_f^t & [\text{PC}] \\ = ok \land \mathbf{CSP1}(\mathbf{R1}(\mathbf{R2}(P \Rightarrow Q))) & [\text{Lemma J.8}] \end{array}$$

Lemma J.19

$$(P;\ Q)_f^t = \boldsymbol{CSP1}\left(\begin{array}{l} (wait' \land P_{post}) \\ \lor ((okay' \land \neg\, wait' \land P_{post});\ Q_{post}) \end{array}\right)$$

provided

1. P and Q are divergence-free
2. $P = \mathbf{R}(P_{pre} \vdash P_{post})$ and $Q = \mathbf{R}(Q_{pre} \vdash Q_{post})$
3. P_{pre} does not mention any dashed variable
4. P_{post} and Q_{post} are **R1** and **R2**

Proof.

$$
\begin{array}{ll}
(P;\ Q)^t_f & \text{[Assumption 2]} \\
= (\mathbf{R}(P_{pre} \vdash P_{post});\ \mathbf{R}(Q_{pre} \vdash Q_{post}))^t_f & \text{[Assumption 1]} \\
= (\mathbf{R}(true \vdash P_{post});\ \mathbf{R}(true \vdash Q_{post}))^t_f & \\
& \text{[Lemma J.7 (Assumptions 3, and 4)]} \\
= \left(\mathbf{R} \left(\begin{array}{l} \left(\begin{array}{l} true \land \\ \neg \left(\begin{array}{l} (okay' \land \neg\ wait' \land P_{post}); \\ \neg\ true \end{array} \right) \end{array} \right) \\ \vdash \\ \left(\begin{array}{l} (wait' \land P_{post}) \\ \lor ((okay' \land \neg\ wait' \land P_{post});\ Q_{post}) \end{array} \right) \end{array} \right) \right)^t_f & \\
& \text{[Sequence and PC]} \\
= \left(\mathbf{R} \left(\begin{array}{l} true \vdash \\ \left(\begin{array}{l} (wait' \land P_{post}) \\ \lor ((okay' \land \neg\ wait' \land P_{post});\ Q_{post}) \end{array} \right) \end{array} \right) \right)^t_f & \\
& \text{[Lemma J.8 and PC]} \\
= \mathbf{CSP1} \left(\mathbf{R1} \left(\mathbf{R2} \left(\begin{array}{l} (wait' \land P_{post}) \\ \lor ((okay' \land \neg\ wait' \land P_{post});\ Q_{post}) \end{array} \right) \right) \right) & \\
& \text{[Lemma J.10 and UTP Laws J.4 and J.5 (Assumption 4)]} \\
= \mathbf{CSP1} \left(\mathbf{R1} \left(\begin{array}{l} (wait' \land P_{post}) \\ \lor ((okay' \land \neg\ wait' \land P_{post});\ Q_{post}) \end{array} \right) \right) & \\
& \text{[UTP Laws J.3, J.1 and J.2 (Assumption 4)]} \\
= \mathbf{CSP1} \left(\begin{array}{l} (wait' \land P_{post}) \\ \lor ((okay' \land \neg\ wait' \land P_{post});\ Q_{post}) \end{array} \right) &
\end{array}
$$

Lemma J.20 *For every **R2** predicate A*

$$\{tr' - tr \mid P[tr \frown s/tr]\} = \{s \frown (tr' - tr) \mid P\}$$

Proof.

$$
\begin{array}{ll}
\{tr' - tr \mid P[tr \frown s/tr]\} & \text{[Notation]} \\
= \{tr' - tr \mid P(tr, tr')[tr \frown s/tr]\} & [\mathbf{R2}] \\
= \{tr' - tr \mid P(\langle\rangle, tr' - tr)[tr \frown s/tr]\} & \text{[Substitution]} \\
= \{tr' - tr \mid P(\langle\rangle, tr' - (tr \frown s))\} & \text{[SC and Sequences]}
\end{array}
$$

$$= \{s \frown (tr' - tr) \mid P\}$$

Lemma J.21

$$\begin{array}{l}\{tr' - tr \mid okay \land \neg\,(tr' = tr \land wait') \land (A)^t_f\} \\ = \{tr' - tr \mid okay \land (A)^t_f\}\end{array}$$

Proof.

$$\begin{array}{ll}
\{tr' - tr \mid okay \land \neg\,(tr' = tr \land wait') \land (A)^t_f\} & [\text{Cases } (wait')] \\
= \{tr' - tr \mid okay \land \neg\,(tr' = tr \land true) \land (A)^t_f\} & [\text{PC and SC}] \\
\quad \cup \{tr' - tr \mid okay \land \neg\,(tr' = tr \land false) \land (A)^t_f\} & \\
= \{tr' - tr \mid okay \land \neg\, tr' = tr \land (A)^t_f\} & [\text{Cases } (tr' = tr)] \\
\quad \cup \{tr' - tr \mid okay \land A)^t_f\} & \\
= \{tr' - tr \mid okay \land \neg\, true \land (A)^t_f\} & [\text{PC and SC}] \\
\quad \cup \{tr' - tr \mid okay \land \neg\, false \land (A)^t_f\} & \\
\quad \cup \{tr' - tr \mid okay \land A)^t_f\} & \\
= \{tr' - tr \mid okay \land (A)^t_f\} &
\end{array}$$

Lemma J.22

$$\begin{array}{l}
\{(\langle\rangle, X) \mid (\langle\rangle, X) \in failures(\Upsilon(A)) \cap failures(\Upsilon(B))\} \\
= \\
\{(\langle\rangle, ref') \mid tr' = tr \land okay \land (A)^t_f \land (B)^t_f\} \\
\cup \{(\langle\rangle, ref' \cup \{\checkmark\}) \mid tr' = tr \land okay \land wait' \land (A)^t_f \land (B)^t_f\}
\end{array}$$

provided

1. *A and B are $\boldsymbol{R}$*
2. *A and B are divergence-free*
3. *$failures^{\mathcal{UTP}}(A) = failures(\Upsilon(A))$*
4. *$failures^{\mathcal{UTP}}(B) = failures(\Upsilon(B))$*

Proof.

$$\{(\langle\rangle, X) \mid (\langle\rangle, X) \in failures(\Upsilon(A)) \cap failures(\Upsilon(B))\} \qquad [\text{Provisos 3 and 4}]$$

$$= \{(\langle\rangle, X) \mid (\langle\rangle, X) \in \mathit{failures}^{\mathcal{UTP}}(A) \cap \mathit{failures}^{\mathcal{UTP}}(B)\}$$

$$[\mathit{failures}^{\mathcal{UTP}}]$$

$$= \left\{ \begin{array}{l} (\langle\rangle, X) \mid \\ (\langle\rangle, X) \in \left(\begin{array}{l} \{(tr' - tr, ref') \mid (A)^n\} \\ \cup \{(tr' - tr, ref' \cup \{\checkmark\}) \mid (A)^n \wedge wait'\} \\ \cup \{((tr' - tr) \frown \langle\checkmark\rangle, ref') \mid (A)^t\} \\ \cup \{((tr' - tr) \frown \langle\checkmark\rangle, ref' \cup \{\checkmark\}) \mid (A)^t\} \end{array} \right) \\ \qquad \cap \left(\begin{array}{l} \{(tr' - tr, ref') \mid (B)^n\} \\ \cup \{(tr' - tr, ref' \cup \{\checkmark\}) \mid (B)^n \wedge wait'\} \\ \cup \{((tr' - tr) \frown \langle\checkmark\rangle, ref') \mid (B)^t\} \\ \cup \{((tr' - tr) \frown \langle\checkmark\rangle, ref' \cup \{\checkmark\}) \mid (B)^t\} \end{array} \right) \end{array} \right\} \qquad [A^t]$$

$$= \left\{ \begin{array}{l} (\langle\rangle, X) \\ \mid (\langle\rangle, X) \in \\ \qquad \left(\begin{array}{l} \{(tr' - tr, ref') \mid (A)^n\} \\ \cup \{(tr' - tr, ref' \cup \{\checkmark\}) \mid (A)^n \wedge wait'\} \\ \cup \{((tr' - tr) \frown \langle\checkmark\rangle, ref') \mid \neg\, wait' \wedge (A)^n\} \\ \cup \{((tr' - tr) \frown \langle\checkmark\rangle, ref' \cup \{\checkmark\}) \mid \neg\, wait' \wedge (A)^n\} \end{array} \right) \\ \qquad \cap \left(\begin{array}{l} \{(tr' - tr, ref') \mid (B)^n\} \\ \cup \{(tr' - tr, ref' \cup \{\checkmark\}) \mid (B)^n \wedge wait'\} \\ \cup \{((tr' - tr) \frown \langle\checkmark\rangle, ref') \mid \neg\, wait' \wedge (B)^n\} \\ \cup \{((tr' - tr) \frown \langle\checkmark\rangle, ref' \cup \{\checkmark\}) \mid \neg\, wait' \wedge (B)^n\} \end{array} \right) \end{array} \right\}$$

$$[A^n]$$

$$
= \left\{ \begin{array}{l}
(\langle\rangle, X) \\
\mid (\langle\rangle, X) \in \\
\qquad \left(\begin{array}{l}
\left\{ \begin{array}{l} (tr' - tr, ref') \\ \mid okay \wedge \neg\, wait \wedge okay' \wedge A \end{array} \right\} \\
\cup \left\{ \begin{array}{l} (tr' - tr, ref' \cup \{\checkmark\}) \\ \mid okay \wedge \neg\, wait \wedge okay' \wedge wait' \wedge A \end{array} \right\} \\
\cup \left\{ \begin{array}{l} ((tr' - tr) \frown \langle\checkmark\rangle, ref') \\ \mid okay \wedge \neg\, wait \wedge okay' \wedge \neg\, wait' \wedge A \end{array} \right\} \\
\cup \left\{ \begin{array}{l} ((tr' - tr) \frown \langle\checkmark\rangle, ref' \cup \{\checkmark\}) \\ \mid okay \wedge \neg\, wait \wedge okay' \wedge \neg\, wait' \wedge A \end{array} \right\}
\end{array} \right) \\
\qquad \cap \left(\begin{array}{l}
\left\{ \begin{array}{l} (tr' - tr, ref') \\ \mid okay \wedge \neg\, wait \wedge okay' \wedge B \end{array} \right\} \\
\cup \left\{ \begin{array}{l} (tr' - tr, ref' \cup \{\checkmark\}) \\ \mid okay \wedge \neg\, wait \wedge okay' \wedge wait' \wedge B \end{array} \right\} \\
\cup \left\{ \begin{array}{l} ((tr' - tr) \frown \langle\checkmark\rangle, ref') \\ \mid okay \wedge \neg\, wait \wedge okay' \wedge \neg\, wait' \wedge B \end{array} \right\} \\
\cup \left\{ \begin{array}{l} ((tr' - tr) \frown \langle\checkmark\rangle, ref' \cup \{\checkmark\}) \\ \mid okay \wedge \neg\, wait \wedge okay' \wedge \neg\, wait' \wedge B \end{array} \right\}
\end{array} \right)
\end{array} \right\}
$$

[PC]

$$
= \left\{ \begin{array}{l}
(\langle\rangle, X) \\
\mid (\langle\rangle, X) \in \\
\qquad \left(\begin{array}{l}
\{(tr' - tr, ref') \mid okay \wedge (A)^t_f\} \\
\cup \{(tr' - tr, ref' \cup \{\checkmark\}) \mid okay \wedge wait' \wedge (A)^t_f\} \\
\cup \{((tr' - tr) \frown \langle\checkmark\rangle, ref') \mid okay \wedge \neg\, wait' \wedge (A)^t_f\} \\
\cup \{((tr' - tr) \frown \langle\checkmark\rangle, ref' \cup \{\checkmark\}) \mid okay \wedge \neg\, wait' \wedge (A)^t_f\}
\end{array} \right) \\
\qquad \cap \left(\begin{array}{l}
\{(tr' - tr, ref') \mid okay \wedge (B)^t_f\} \\
\cup \{(tr' - tr, ref' \cup \{\checkmark\}) \mid okay \wedge wait' \wedge (B)^t_f\} \\
\cup \{((tr' - tr) \frown \langle\checkmark\rangle, ref') \mid okay \wedge \neg\, wait' \wedge (B)^t_f\} \\
\cup \{((tr' - tr) \frown \langle\checkmark\rangle, ref' \cup \{\checkmark\}) \mid okay \wedge \neg\, wait' \wedge (B)^t_f\}
\end{array} \right)
\end{array} \right\}
$$

[ST and SC $(tick \notin \operatorname{ran}(tr') \cup \operatorname{ran}(tr) \cup ref')$]

$$= \left\{ \begin{array}{l} (\langle\rangle, X) \\ \mid (\langle\rangle, X) \in \\ \qquad \left\{ \begin{array}{l} (tr' - tr, ref') \\ \mid okay \wedge (A)^t_f \wedge (B)^t_f \end{array} \right\} \\ \qquad \cup \left\{ \begin{array}{l} (tr' - tr, ref' \cup \{\checkmark\}) \\ \mid okay \wedge wait' \wedge (A)^t_f \wedge (B)^t_f \end{array} \right\} \\ \qquad \cup \left\{ \begin{array}{l} ((tr' - tr) ^\frown \langle\checkmark\rangle, ref') \\ \mid okay \wedge \neg\, wait' \wedge (A)^t_f \wedge (B)^t_f \end{array} \right\} \\ \qquad \cup \left\{ \begin{array}{l} ((tr' - tr) ^\frown \langle\checkmark\rangle, ref' \cup \{\checkmark\}) \\ \mid okay \wedge \neg\, wait' \wedge (A)^t_f \wedge (B)^t_f \end{array} \right\} \end{array} \right\}$$

[SC and $-$]

$$\begin{array}{l} = \{(\langle\rangle, ref') \mid tr' = tr \wedge okay \wedge (A)^t_f \wedge (B)^t_f\} \\ \quad \cup \{(\langle\rangle, ref' \cup \{\checkmark\}) \mid tr' = tr \wedge okay \wedge wait' \wedge (A)^t_f \wedge (B)^t_f\} \end{array}$$

Lemma J.23

$$\begin{array}{l} \{(s, X) \mid (s, X) \in failures(\Upsilon(A)) \cup failures(\Upsilon(B)) \wedge s \neq \langle\rangle\} \\ = \\ \{(tr' - tr, ref') \mid tr' = tr \wedge okay \wedge (A)^t_f\} \\ \cup \{(tr' - tr, ref' \cup \{\checkmark\}) \mid tr' = tr \wedge okay \wedge wait' \wedge (A)^t_f\} \\ \cup \{((tr' - tr) ^\frown \langle\checkmark\rangle, ref') \mid okay \wedge \neg\, wait' \wedge (A)^t_f\} \\ \cup \{((tr' - tr) ^\frown \langle\checkmark\rangle, ref' \cup \{\checkmark\}) \mid okay \wedge \neg\, wait' \wedge (A)^t_f\} \\ \cup \{(tr' - tr, ref') \mid tr' = tr \wedge okay \wedge (B)^t_f\} \\ \cup \{(tr' - tr, ref' \cup \{\checkmark\}) \mid tr' = tr \wedge okay \wedge wait' \wedge (B)^t_f\} \\ \cup \{((tr' - tr) ^\frown \langle\checkmark\rangle, ref') \mid okay \wedge \neg\, wait' \wedge (B)^t_f\} \\ \cup \{((tr' - tr) ^\frown \langle\checkmark\rangle, ref' \cup \{\checkmark\}) \mid okay \wedge \neg\, wait' \wedge (B)^t_f\} \end{array}$$

provided

1. *A and B are $\boldsymbol{R}$*
2. *A and B are divergence-free*
3. *$failures^{\mathcal{UTP}}(A) = failures(\Upsilon(A))$*
4. *$failures^{\mathcal{UTP}}(B) = failures(\Upsilon(B))$*

Proof.

$$\{(s, X) \mid (s, X) \in failures(\Upsilon(A)) \cup failures(\Upsilon(B)) \wedge s \neq \langle\rangle\} \quad \text{[Provisos 3 and 4]}$$
$$= \{(s, X) \mid (s, X) \in failures^{\mathcal{UTP}}(A) \cup failures^{\mathcal{UTP}}(B) \wedge s \neq \langle\rangle\}$$

[$failures^{\mathcal{UTP}}$]

$$= \left\{ \begin{array}{l} (s, X) \\ \mid s \neq \langle\rangle \\ \land (s, X) \in \left(\begin{array}{l} \{(tr' - tr, ref') \mid (A)^n\} \\ \cup \{(tr' - tr, ref' \cup \{\checkmark\}) \mid (A)^n \land wait'\} \\ \cup \{((tr' - tr) \frown \langle\checkmark\rangle, ref') \mid (A)^t\} \\ \cup \{((tr' - tr) \frown \langle\checkmark\rangle, ref' \cup \{\checkmark\}) \mid (A)^t\} \end{array} \right) \\ \qquad \cup \left(\begin{array}{l} \{(tr' - tr, ref') \mid (B)^n\} \\ \cup \{(tr' - tr, ref' \cup \{\checkmark\}) \mid (B)^n \land wait'\} \\ \cup \{((tr' - tr) \frown \langle\checkmark\rangle, ref') \mid (B)^t\} \\ \cup \{((tr' - tr) \frown \langle\checkmark\rangle, ref' \cup \{\checkmark\}) \mid (B)^t\} \end{array} \right) \end{array} \right\} \qquad [A^t]$$

$$= \left\{ \begin{array}{l} (s, X) \\ \mid s \neq \langle\rangle \\ \land (s, X) \in \left(\begin{array}{l} \{(tr' - tr, ref') \mid (A)^n\} \\ \cup \{(tr' - tr, ref' \cup \{\checkmark\}) \mid (A)^n \land wait'\} \\ \cup \{((tr' - tr) \frown \langle\checkmark\rangle, ref') \mid \neg\, wait' \land (A)^n\} \\ \cup \{((tr' - tr) \frown \langle\checkmark\rangle, ref' \cup \{\checkmark\}) \mid \neg\, wait' \land (A)^n\} \end{array} \right) \\ \qquad \cup \left(\begin{array}{l} \{(tr' - tr, ref') \mid (B)^n\} \\ \cup \{(tr' - tr, ref' \cup \{\checkmark\}) \mid (B)^n \land wait'\} \\ \cup \{((tr' - tr) \frown \langle\checkmark\rangle, ref') \mid \neg\, wait' \land (B)^n\} \\ \cup \{((tr' - tr) \frown \langle\checkmark\rangle, ref' \cup \{\checkmark\}) \mid \neg\, wait' \land (B)^n\} \end{array} \right) \end{array} \right\}$$

$$[A^n]$$

$$= \left\{ \begin{array}{l} (s, X) \\ \mid s \neq \langle\rangle \\ \land (s, X) \in \left(\begin{array}{l} \left\{ \begin{array}{l} (tr' - tr, ref') \\ \mid okay \land \neg\, wait \land okay' \land A \end{array} \right\} \\ \cup \left\{ \begin{array}{l} (tr' - tr, ref' \cup \{\checkmark\}) \\ \mid okay \land \neg\, wait \land okay' \land wait' \land A \end{array} \right\} \\ \cup \left\{ \begin{array}{l} ((tr' - tr) \frown \langle\checkmark\rangle, ref') \\ \mid okay \land \neg\, wait \land okay' \land \neg\, wait' \land A \end{array} \right\} \\ \cup \left\{ \begin{array}{l} ((tr' - tr) \frown \langle\checkmark\rangle, ref' \cup \{\checkmark\}) \\ \mid okay \land \neg\, wait \land okay' \land \neg\, wait' \land A \end{array} \right\} \end{array} \right) \\ \qquad \cup \left(\begin{array}{l} \left\{ \begin{array}{l} (tr' - tr, ref') \\ \mid okay \land \neg\, wait \land okay' \land B \end{array} \right\} \\ \cup \left\{ \begin{array}{l} (tr' - tr, ref' \cup \{\checkmark\}) \\ \mid okay \land \neg\, wait \land okay' \land wait' \land B \end{array} \right\} \\ \cup \left\{ \begin{array}{l} ((tr' - tr) \frown \langle\checkmark\rangle, ref') \\ \mid okay \land \neg\, wait \land okay' \land \neg\, wait' \land B \end{array} \right\} \\ \cup \left\{ \begin{array}{l} ((tr' - tr) \frown \langle\checkmark\rangle, ref' \cup \{\checkmark\}) \\ \mid okay \land \neg\, wait \land okay' \land \neg\, wait' \land B \end{array} \right\} \end{array} \right) \end{array} \right\}$$

[PC]

$$= \left\{ \begin{array}{l} (s, X) \\ \mid s \neq \langle\rangle \wedge (s, X) \in \\ \qquad \left(\begin{array}{l} \{(tr' - tr, ref') \mid okay \wedge (A)^t_f\} \\ \cup \{(tr' - tr, ref' \cup \{\checkmark\}) \mid okay \wedge wait' \wedge (A)^t_f\} \\ \cup \{((tr' - tr) \frown \langle\checkmark\rangle, ref') \mid okay \wedge \neg\ wait' \wedge (A)^t_f\} \\ \cup \{((tr' - tr) \frown \langle\checkmark\rangle, ref' \cup \{\checkmark\}) \mid okay \wedge \neg\ wait' \wedge (A)^t_f\} \end{array} \right) \\ \qquad \cup \left(\begin{array}{l} \{(tr' - tr, ref') \mid okay \wedge (B)^t_f\} \\ \cup \{(tr' - tr, ref' \cup \{\checkmark\}) \mid okay \wedge wait' \wedge (B)^t_f\} \\ \cup \{((tr' - tr) \frown \langle\checkmark\rangle, ref') \mid okay \wedge \neg\ wait' \wedge (B)^t_f\} \\ \cup \{((tr' - tr) \frown \langle\checkmark\rangle, ref' \cup \{\checkmark\}) \mid okay \wedge \neg\ wait' \wedge (B)^t_f\} \end{array} \right) \end{array} \right\}$$

[ST and SC]

$$\begin{array}{l} = \{(tr' - tr, ref') \mid tr' - tr \neq \langle\rangle \wedge okay \wedge (A)^t_f\} \\ \quad \cup \{(tr' - tr, ref' \cup \{\checkmark\}) \mid tr' - tr \neq \langle\rangle \wedge okay \wedge wait' \wedge (A)^t_f\} \\ \quad \cup \{((tr' - tr) \frown \langle\checkmark\rangle, ref') \mid okay \wedge \neg\ wait' \wedge (A)^t_f\} \\ \quad \cup \{((tr' - tr) \frown \langle\checkmark\rangle, ref' \cup \{\checkmark\}) \mid okay \wedge \neg\ wait' \wedge (A)^t_f\} \\ \quad \cup \{(tr' - tr, ref') \mid tr' - tr \neq \langle\rangle \wedge okay \wedge (B)^t_f\} \\ \quad \cup \{(tr' - tr, ref' \cup \{\checkmark\}) \mid tr' - tr \neq \langle\rangle \wedge okay \wedge wait' \wedge (B)^t_f\} \\ \quad \cup \{((tr' - tr) \frown \langle\checkmark\rangle, ref') \mid okay \wedge \neg\ wait' \wedge (B)^t_f\} \\ \quad \cup \{((tr' - tr) \frown \langle\checkmark\rangle, ref' \cup \{\checkmark\}) \mid okay \wedge \neg\ wait' \wedge (B)^t_f\} \end{array}$$

[−]

$$\begin{array}{l} = \{(tr' - tr, ref') \mid tr' = tr \wedge okay \wedge (A)^t_f\} \\ \quad \cup \{(tr' - tr, ref' \cup \{\checkmark\}) \mid tr' = tr \wedge okay \wedge wait' \wedge (A)^t_f\} \\ \quad \cup \{((tr' - tr) \frown \langle\checkmark\rangle, ref') \mid okay \wedge \neg\ wait' \wedge (A)^t_f\} \\ \quad \cup \{((tr' - tr) \frown \langle\checkmark\rangle, ref' \cup \{\checkmark\}) \mid okay \wedge \neg\ wait' \wedge (A)^t_f\} \\ \quad \cup \{(tr' - tr, ref') \mid tr' = tr \wedge okay \wedge (B)^t_f\} \\ \quad \cup \{(tr' - tr, ref' \cup \{\checkmark\}) \mid tr' = tr \wedge okay \wedge wait' \wedge (B)^t_f\} \\ \quad \cup \{((tr' - tr) \frown \langle\checkmark\rangle, ref') \mid okay \wedge \neg\ wait' \wedge (B)^t_f\} \\ \quad \cup \{((tr' - tr) \frown \langle\checkmark\rangle, ref' \cup \{\checkmark\}) \mid okay \wedge \neg\ wait' \wedge (B)^t_f\} \end{array}$$

Lemma J.24

$$\begin{array}{l} \{(\langle\rangle, X) \mid X \subseteq \Sigma \wedge \langle\checkmark\rangle \in traces(\Upsilon(A \Box B))\} \\ = \\ \{(\langle\rangle, ref') \mid tr' = tr \wedge okay \wedge \neg\ wait' \wedge (A)^t_f\}\} \\ \cup \{(\langle\rangle, ref') \mid tr' = tr \wedge okay \wedge \neg\ wait' \wedge (B)^t_f\}\} \end{array}$$

provided

1. *A and B are* ***R***

2. A and B are divergence-free

Proof.

$$\{(\langle\rangle, X) \mid X \subseteq \Sigma \land \langle\checkmark\rangle \in traces(\Upsilon(A \Box B))\} \qquad \text{[Theorem J.10 (Provisos)]}$$

$$= \{(\langle\rangle, X) \mid X \subseteq \Sigma \land \langle\checkmark\rangle \in traces^{\mathcal{UTP}}(A \Box B)\} \qquad [traces^{\mathcal{UTP}}]$$

$$= \left\{ \begin{array}{l} (\langle\rangle, X) \\ \mid X \subseteq \Sigma \land \\ \qquad \langle\checkmark\rangle \in \left\{ \begin{array}{l} \{tr' - tr \mid (A \Box B)^n\} \\ \cup \{(tr' - tr) \frown \langle\checkmark\rangle \mid (A \Box B)^t\} \end{array} \right\} \end{array} \right\} \qquad [A^t]$$

$$= \left\{ \begin{array}{l} (\langle\rangle, X) \\ \mid X \subseteq \Sigma \land \\ \qquad \langle\checkmark\rangle \in \{tr' - tr \mid (A \Box B)^n\} \\ \qquad\qquad \cup \{(tr' - tr) \frown \langle\checkmark\rangle \mid \neg\, wait' \land (A \Box B)^n\} \end{array} \right\} \qquad [A^n]$$

$$= \left\{ \begin{array}{l} (\langle\rangle, X) \mid X \subseteq \Sigma \land \\ \qquad \langle\checkmark\rangle \in \left\{ \begin{array}{l} tr' - tr \mid okay \land \neg\, wait \land okay' \land \\ \qquad (A \Box B) \end{array} \right\} \\ \qquad\qquad \cup \left\{ \begin{array}{l} (tr' - tr) \frown \langle\checkmark\rangle \mid okay \land \neg\, wait \land okay' \land \neg\, wait' \land \\ \qquad A \Box B \end{array} \right\} \end{array} \right\} \qquad [PC]$$

$$= \left\{ \begin{array}{l} (\langle\rangle, X) \mid X \subseteq \Sigma \land \\ \qquad \langle\checkmark\rangle \in \left\{ \begin{array}{l} tr' - tr \\ \mid okay \land (A \Box B)^t_f \end{array} \right\} \\ \qquad\qquad \cup \left\{ \begin{array}{l} (tr' - tr) \frown \langle\checkmark\rangle \\ \mid okay \land \neg\, wait' \land (A \Box B)^t_f \end{array} \right\} \end{array} \right\} \qquad \text{[Lemma J.17]}$$

$$
= \left\{ \begin{array}{l}
(\langle\rangle, X) \mid X \subseteq \Sigma \land \\
\langle \checkmark \rangle \in \left\{ \begin{array}{l}
tr' - tr \\
\mid okay \land \\
\mathbf{CSP1} \left(\begin{array}{l}
(\neg\, (A)^f_f \land \neg\, (B)^f_f) \\
\Rightarrow \\
\left(\begin{array}{l}
((A)^t_f \land (B)^t_f) \\
\lhd tr' = tr \land wait' \rhd \\
((A)^t_f \lor (B)^t_f)
\end{array} \right)
\end{array} \right)
\end{array} \right\} \\
\cup \left\{ \begin{array}{l}
(tr' - tr) ^\frown \langle \checkmark \rangle \\
\mid okay \land \neg\, wait' \land \\
\mathbf{CSP1} \left(\begin{array}{l}
(\neg\, (A)^f_f \land \neg\, (B)^f_f) \\
\Rightarrow \\
\left(\begin{array}{l}
((A)^t_f \land (B)^t_f) \\
\lhd tr' = tr \land wait' \rhd \\
((A)^t_f \lor (B)^t_f)
\end{array} \right)
\end{array} \right)
\end{array} \right\}
\end{array} \right\}
$$

[Lemma J.4]

$$
= \left\{ \begin{array}{l}
(\langle\rangle, X) \mid X \subseteq \Sigma \land \\
\langle \checkmark \rangle \in \left\{ \begin{array}{l}
tr' - tr \\
\mid okay \land \\
\left(\begin{array}{l}
(\neg\, (A)^f_f \land \neg\, (B)^f_f) \\
\Rightarrow \\
\left(\begin{array}{l}
((A)^t_f \land (B)^t_f) \\
\lhd tr' = tr \land wait' \rhd \\
((A)^t_f \lor (B)^t_f)
\end{array} \right)
\end{array} \right)
\end{array} \right\} \\
\cup \left\{ \begin{array}{l}
(tr' - tr) ^\frown \langle \checkmark \rangle \\
\mid okay \land \neg\, wait' \land \\
\left(\begin{array}{l}
(\neg\, (A)^f_f \land \neg\, (B)^f_f) \\
\Rightarrow \\
\left(\begin{array}{l}
((A)^t_f \land (B)^t_f) \\
\lhd tr' = tr \land wait' \rhd \\
((A)^t_f \lor (B)^t_f)
\end{array} \right)
\end{array} \right)
\end{array} \right\}
\end{array} \right\}
$$

[Proviso 2 and PC]

$$= \left\{ \begin{array}{l} (\langle\rangle, X) \mid X \subseteq \Sigma \land \\ \qquad \langle \checkmark \rangle \in \left\{ \begin{array}{l} tr' - tr \\ \mid okay \land \\ \left(\begin{array}{l} ((A)^t_f \land (B)^t_f) \\ \lhd tr' = tr \land wait' \rhd \\ ((A)^t_f \lor (B)^t_f) \end{array} \right) \end{array} \right\} \\ \qquad \cup \left\{ \begin{array}{l} (tr' - tr) \frown \langle \checkmark \rangle \\ \mid okay \land \neg\, wait' \land \\ \left(\begin{array}{l} ((A)^t_f \land (B)^t_f) \\ \lhd tr' = tr \land wait' \rhd \\ ((A)^t_f \lor (B)^t_f) \end{array} \right) \end{array} \right\} \end{array} \right\}$$

[PC and SC]

$$= \left\{ \begin{array}{l} (\langle\rangle, X) \mid X \subseteq \Sigma \land \\ \qquad \langle \checkmark \rangle \in \left\{ \begin{array}{l} tr' - tr \\ \mid okay \land tr' = tr \land wait' \land (A)^t_f \land (B)^t_f \end{array} \right\} \\ \qquad \cup \left\{ \begin{array}{l} tr' - tr \\ \mid okay \land \neg\,(tr' = tr \land wait') \land (A)^t_f \end{array} \right\} \\ \qquad \cup \left\{ \begin{array}{l} tr' - tr \\ \mid okay \land \neg\,(tr' = tr \land wait') \land (B)^t_f \end{array} \right\} \\ \qquad \cup \left\{ \begin{array}{l} (tr' - tr) \frown \langle \checkmark \rangle \\ \mid okay \land \neg\, wait' \land (A)^t_f \end{array} \right\} \\ \qquad \cup \left\{ \begin{array}{l} (tr' - tr) \frown \langle \checkmark \rangle \\ \mid okay \land \neg\, wait' \land (B)^t_f \end{array} \right\} \end{array} \right\}$$

[Cases, −, PC and SC]

$$= \left\{ \begin{array}{l} (\langle\rangle, X) \mid X \subseteq \Sigma \land \\ \qquad \langle \checkmark \rangle \in \{\langle\rangle\} \\ \qquad \cup \left\{ \begin{array}{l} tr' - tr \\ \mid okay \land \neg\,(tr' = tr \land wait') \land (A)^t_f \end{array} \right\} \\ \qquad \cup \left\{ \begin{array}{l} tr' - tr \\ \mid okay \land \neg\,(tr' = tr \land wait') \land (B)^t_f \end{array} \right\} \\ \qquad \cup \left\{ \begin{array}{l} (tr' - tr) \frown \langle \checkmark \rangle \\ \mid okay \land \neg\, wait' \land (A)^t_f \end{array} \right\} \\ \qquad \cup \left\{ \begin{array}{l} (tr' - tr) \frown \langle \checkmark \rangle \\ \mid okay \land \neg\, wait' \land (B)^t_f \end{array} \right\} \end{array} \right\}$$

[Lemma J.21]

$$= \left\{ \begin{array}{l} (\langle\rangle, X) \mid X \subseteq \Sigma \wedge \\ \qquad \langle \checkmark \rangle \in \{\langle\rangle\} \\ \qquad\qquad \cup \{tr' - tr \mid okay \wedge (A)^t_f\} \\ \qquad\qquad \cup \{tr' - tr \mid okay \wedge (B)^t_f\} \\ \qquad\qquad \cup \{(tr' - tr) \frown \langle \checkmark \rangle \mid okay \wedge \neg\, wait' \wedge (A)^t_f\} \\ \qquad\qquad \cup \{(tr' - tr) \frown \langle \checkmark \rangle \mid okay \wedge \neg\, wait' \wedge (B)^t_f\} \end{array} \right\}$$

[ST and SC]

$$\begin{array}{l} = \{(\langle\rangle, X) \mid X \subseteq \Sigma \wedge \langle \checkmark \rangle \in \{tr' - tr \mid okay \wedge (A)^t_f\}\} \\ \quad \cup \{(\langle\rangle, X) \mid X \subseteq \Sigma \wedge \langle \checkmark \rangle \in \{tr' - tr \mid okay \wedge (B)^t_f\}\} \\ \quad \cup \{(\langle\rangle, X) \mid X \subseteq \Sigma \wedge \langle \checkmark \rangle \in \{(tr' - tr) \frown \langle \checkmark \rangle \mid okay \wedge \neg\, wait' \wedge (A)^t_f\}\} \\ \quad \cup \{(\langle\rangle, X) \mid X \subseteq \Sigma \wedge \langle \checkmark \rangle \in \{(tr' - tr) \frown \langle \checkmark \rangle \mid okay \wedge \neg\, wait' \wedge (B)^t_f\}\} \end{array}$$

$[ref']$

$$\begin{array}{l} = \{(\langle\rangle, ref') \mid \langle \checkmark \rangle \in \{tr' - tr \mid okay \wedge (A)^t_f\}\} \\ \quad \cup \{(\langle\rangle, ref') \mid \langle \checkmark \rangle \in \{tr' - tr \mid okay \wedge (B)^t_f\}\} \\ \quad \cup \{(\langle\rangle, ref') \mid \langle \checkmark \rangle \in \{(tr' - tr) \frown \langle \checkmark \rangle \mid okay \wedge \neg\, wait' \wedge (A)^t_f\}\} \\ \quad \cup \{(\langle\rangle, ref') \mid \langle \checkmark \rangle \in \{(tr' - tr) \frown \langle \checkmark \rangle \mid okay \wedge \neg\, wait' \wedge (B)^t_f\}\} \end{array}$$

[SC $(tick \notin \mathrm{ran}(tr') \cup \mathrm{ran}(tr))$]

$$\begin{array}{l} = \{(\langle\rangle, ref') \mid \langle \checkmark \rangle \in \{(tr' - tr) \frown \langle \checkmark \rangle \mid okay \wedge \neg\, wait' \wedge (A)^t_f\}\} \\ \quad \cup \{(\langle\rangle, ref') \mid \langle \checkmark \rangle \in \{(tr' - tr) \frown \langle \checkmark \rangle \mid okay \wedge \neg\, wait' \wedge (B)^t_f\}\} \end{array}$$

[SC and $-$]

$$\begin{array}{l} = \{(\langle\rangle, ref') \mid tr' = tr \wedge okay \wedge \neg\, wait' \wedge (A)^t_f\}\} \\ \quad \cup \{(\langle\rangle, ref') \mid tr' = tr \wedge okay \wedge \neg\, wait' \wedge (B)^t_f\}\} \end{array}$$

Theorem J.1

$$\begin{array}{l} \{(\langle\rangle, ref') \mid okay \wedge tr' = tr \wedge wait' \wedge (A)^t_f \wedge (B)^t_f\} \\ \cup \{(\langle\rangle, ref' \cup \{\checkmark\}) \mid okay \wedge wait' \wedge tr' = tr \wedge (A)^t_f \wedge (B)^t_f\} \\ \cup \{(tr' - tr, ref') \mid okay \wedge \neg\, (tr' = tr \wedge wait') \wedge (A)^t_f\} \\ \cup \{(tr' - tr, ref') \mid okay \wedge \neg\, (tr' = tr \wedge wait') \wedge (B)^t_f\} \\ = \\ \{(\langle\rangle, ref') \mid tr' = tr \wedge okay \wedge (A)^t_f \wedge (B)^t_f\} \\ \cup \{(\langle\rangle, ref') \mid tr' = tr \wedge okay \wedge \neg\, wait' \wedge (A)^t_f\}\} \\ \cup \{(\langle\rangle, ref') \mid tr' = tr \wedge okay \wedge \neg\, wait' \wedge (B)^t_f\}\} \\ \cup \{(tr' - tr, ref') \mid \neg\, tr' = tr \wedge okay \wedge (A)^t_f\} \\ \cup \{(tr' - tr, ref') \mid \neg\, tr' = tr \wedge okay \wedge (B)^t_f\} \\ \cup \{(\langle\rangle, ref' \cup \{\checkmark\}) \mid tr' = tr \wedge okay \wedge wait' \wedge (A)^t_f \wedge (B)^t_f\} \end{array}$$

Proof. This proof is achieved by case analysis on the conditions that determines the choice, that is, $tr' = tr \wedge wait'$. Both actions A and B either imply on this condition or not. We have four cases.

Case 1.

$$\begin{array}{l}(A)^t_f \Rightarrow tr' = tr \wedge wait' \\ \wedge \\ (B)^t_f \Rightarrow tr' = tr \wedge wait'\end{array}$$

Proof.

$$\begin{array}{l}
\{(\langle\rangle, ref') \mid okay \wedge tr' = tr \wedge wait' \wedge (A)^t_f \wedge (B)^t_f\} \\
\cup \{(\langle\rangle, ref' \cup \{\checkmark\}) \mid okay \wedge wait' \wedge tr' = tr \wedge (A)^t_f \wedge (B)^t_f\} \\
\cup \{(tr' - tr, ref') \mid okay \wedge \neg\, (tr' = tr \wedge wait') \wedge (A)^t_f\} \\
\cup \{(tr' - tr, ref') \mid okay \wedge \neg\, (tr' = tr \wedge wait') \wedge (B)^t_f\}
\end{array}$$

[Assumption and PC]

$$\begin{array}{l}
= \{(\langle\rangle, ref') \mid okay \wedge true \wedge (A)^t_f \wedge (B)^t_f\} \\
\quad \cup \{(\langle\rangle, ref' \cup \{\checkmark\}) \mid okay \wedge true \wedge (A)^t_f \wedge (B)^t_f\} \\
\quad \cup \{(tr' - tr, ref') \mid okay \wedge false \wedge (A)^t_f\} \\
\quad \cup \{(tr' - tr, ref') \mid okay \wedge false \wedge (B)^t_f\}
\end{array}$$

[PC, SC, and ST]

$$\begin{array}{l}
= \{(\langle\rangle, ref') \mid okay \wedge (A)^t_f \wedge (B)^t_f\} \\
\quad \cup \{(\langle\rangle, ref' \cup \{\checkmark\}) \mid okay \wedge (A)^t_f \wedge (B)^t_f\}
\end{array}$$

[PC, SC, and ST]

$$\begin{array}{l}
= \{(\langle\rangle, ref') \mid okay \wedge (A)^t_f \wedge (B)^t_f\} \\
\quad \cup \{(\langle\rangle, ref') \mid okay \wedge false \wedge (A)^t_f\}\} \\
\quad \cup \{(\langle\rangle, ref') \mid okay \wedge false \wedge (B)^t_f\}\} \\
\quad \cup \{(tr' - tr, ref') \mid false \wedge okay \wedge (A)^t_f\} \\
\quad \cup \{(tr' - tr, ref') \mid false \wedge okay \wedge (B)^t_f\} \\
\quad \cup \{(\langle\rangle, ref' \cup \{\checkmark\}) \mid okay \wedge true \wedge (A)^t_f \wedge (B)^t_f\}
\end{array}$$

[Assumption and PC]

$$\begin{array}{l}
= \{(\langle\rangle, ref') \mid tr' = tr \wedge okay \wedge (A)^t_f \wedge (B)^t_f\} \\
\quad \cup \{(\langle\rangle, ref') \mid tr' = tr \wedge okay \wedge \neg\, wait' \wedge (A)^t_f\}\} \\
\quad \cup \{(\langle\rangle, ref') \mid tr' = tr \wedge okay \wedge \neg\, wait' \wedge (B)^t_f\}\} \\
\quad \cup \{(tr' - tr, ref') \mid \neg\, tr' = tr \wedge okay \wedge (A)^t_f\} \\
\quad \cup \{(tr' - tr, ref') \mid \neg\, tr' = tr \wedge okay \wedge (B)^t_f\} \\
\quad \cup \{(\langle\rangle, ref' \cup \{\checkmark\}) \mid tr' = tr \wedge okay \wedge wait' \wedge (A)^t_f \wedge (B)^t_f\}
\end{array}$$

Case 2.

$$\begin{aligned}
&(A)^t_f \Rightarrow \neg\,(tr' = tr \wedge wait')\\
&\wedge\\
&(B)^t_f \Rightarrow tr' = tr \wedge wait'
\end{aligned}$$

Proof.

$$\begin{aligned}
&\{(\langle\rangle, ref') \mid okay \wedge tr' = tr \wedge wait' \wedge (A)^t_f \wedge (B)^t_f\}\\
&\cup \{(\langle\rangle, ref' \cup \{\checkmark\}) \mid okay \wedge wait' \wedge tr' = tr \wedge (A)^t_f \wedge (B)^t_f\}\\
&\cup \{(tr' - tr, ref') \mid okay \wedge \neg\,(tr' = tr \wedge wait') \wedge (A)^t_f\}\\
&\cup \{(tr' - tr, ref') \mid okay \wedge \neg\,(tr' = tr \wedge wait') \wedge (B)^t_f\}
\end{aligned}$$

[Assumption and PC]

$$\begin{aligned}
&= \{(\langle\rangle, ref') \mid okay \wedge false \wedge (A)^t_f \wedge (B)^t_f\}\\
&\quad \cup \{(\langle\rangle, ref' \cup \{\checkmark\}) \mid okay \wedge false \wedge (A)^t_f \wedge (B)^t_f\}\\
&\quad \cup \{(tr' - tr, ref') \mid okay \wedge true \wedge (A)^t_f\}\\
&\quad \cup \{(tr' - tr, ref') \mid okay \wedge false \wedge (B)^t_f\}
\end{aligned}$$

[PC, SC, and ST]

$$= \{(tr' - tr, ref') \mid okay \wedge (A)^t_f\} \qquad \text{[Assumption, PC, SC, and ST]}$$

$$\begin{aligned}
&= \{(tr' - tr, ref') \mid okay \wedge \neg\, tr' = tr \wedge (A)^t_f\} && \text{[Cases on } tr' = tr\text{, PC, SC, ST]}\\
&\quad \cup \{(tr' - tr, ref') \mid okay \wedge \neg\, wait' \wedge (A)^t_f\}
\end{aligned}$$

$$\begin{aligned}
&= \{(tr' - tr, ref') \mid okay \wedge \neg\, tr' = tr \wedge (A)^t_f\} && \text{[}-\text{, PC and ST]}\\
&\quad \cup \{(tr' - tr, ref') \mid okay \wedge \neg\, wait' \wedge tr' = tr \wedge (A)^t_f\}\\
&\quad \cup \{(tr' - tr, ref') \mid okay \wedge \neg\, wait' \wedge \neg\, tr' = tr \wedge (A)^t_f\}
\end{aligned}$$

$$\begin{aligned}
&= \{(\langle\rangle, ref') \mid okay \wedge tr' = tr \wedge \neg\, wait' \wedge (A)^t_f\} && \text{[SC and ST]}\\
&\quad \cup \{(tr' - tr, ref') \mid okay \wedge \neg\, wait' \wedge \neg\, tr' = tr \wedge (A)^t_f\}\\
&\quad \cup \{(tr' - tr, ref') \mid okay \wedge \neg\, tr' = tr \wedge (A)^t_f\}
\end{aligned}$$

$$\begin{aligned}
&= \{(\langle\rangle, ref') \mid okay \wedge tr' = tr \wedge \neg\, wait' \wedge (A)^t_f\}\} && \text{[PC, SC, and ST]}\\
&\quad \cup \{(tr' - tr, ref') \mid okay \wedge \neg\, tr' = tr \wedge (A)^t_f\}
\end{aligned}$$

$$\begin{aligned}
&= \{(\langle\rangle, ref') \mid tr' = tr \wedge okay \wedge false \wedge (A)^t_f \wedge (B)^t_f\}\\
&\quad \cup \{(\langle\rangle, ref') \mid tr' = tr \wedge okay \wedge \neg\, wait' \wedge (A)^t_f\}\}\\
&\quad \cup \{(\langle\rangle, ref') \mid tr' = tr \wedge okay \wedge false \wedge (B)^t_f\}\}\\
&\quad \cup \{(tr' - tr, ref') \mid \neg\, tr' = tr \wedge okay \wedge (A)^t_f\}\\
&\quad \cup \{(tr' - tr, ref') \mid false \wedge okay \wedge (B)^t_f\}\\
&\quad \cup \{(\langle\rangle, ref' \cup \{\checkmark\}) \mid tr' = tr \wedge okay \wedge wait' \wedge false \wedge (A)^t_f \wedge (B)^t_f\}
\end{aligned}$$

[Assumption and PC]

$$
\begin{aligned}
&= \{(\langle\rangle, ref') \mid tr' = tr \land okay \land (A)^t_f \land (B)^t_f\} \\
&\quad \cup \{(\langle\rangle, ref') \mid tr' = tr \land okay \land \neg\, wait' \land (A)^t_f\}\} \\
&\quad \cup \{(\langle\rangle, ref') \mid tr' = tr \land okay \land \neg\, wait' \land (B)^t_f\}\} \\
&\quad \cup \{(tr' - tr, ref') \mid \neg\, tr' = tr \land okay \land (A)^t_f\} \\
&\quad \cup \{(tr' - tr, ref') \mid \neg\, tr' = tr \land okay \land (B)^t_f\} \\
&\quad \cup \{(\langle\rangle, ref' \cup \{\checkmark\}) \mid tr' = tr \land okay \land wait' \land (A)^t_f \land (B)^t_f\}
\end{aligned}
$$

Case 3.

$$
\begin{aligned}
&(A)^t_f \Rightarrow tr' = tr \land wait' \\
&\land \\
&(B)^t_f \Rightarrow \neg\,(tr' = tr \land wait')
\end{aligned}
$$

Proof. Analogous to Case 2 above.

Case 4.

$$
\begin{aligned}
&(A)^t_f \Rightarrow \neg\,(tr' = tr \land wait') \\
&\land \\
&(B)^t_f \Rightarrow \neg\,(tr' = tr \land wait')
\end{aligned}
$$

Proof.

$$\{(\langle\rangle, ref') \mid okay \land tr' = tr \land wait' \land (A)^t_f \land (B)^t_f\}$$
$$\cup \{(\langle\rangle, ref' \cup \{\checkmark\}) \mid okay \land wait' \land tr' = tr \land (A)^t_f \land (B)^t_f\}$$
$$\cup \{(tr' - tr, ref') \mid okay \land \neg (tr' = tr \land wait') \land (A)^t_f\}$$
$$\cup \{(tr' - tr, ref') \mid okay \land \neg (tr' = tr \land wait') \land (B)^t_f\}$$

[Assumption and PC]

$$\{(\langle\rangle, ref') \mid okay \land false \land (A)^t_f \land (B)^t_f\}$$ [PC, SC, and ST]
$$\cup \{(\langle\rangle, ref' \cup \{\checkmark\}) \mid okay \land false \land (A)^t_f \land (B)^t_f\}$$
$$\cup \{(tr' - tr, ref') \mid okay \land true \land (A)^t_f\}$$
$$\cup \{(tr' - tr, ref') \mid okay \land true \land (B)^t_f\}$$

$$= \{(tr' - tr, ref') \mid okay \land (A)^t_f\}$$ [Assumption, PC, SC, ST]
$$\cup \{(tr' - tr, ref') \mid okay \land (B)^t_f\}$$

$$= \{(tr' - tr, ref') \mid okay \land \neg\, wait' \land (A)^t_f\}$$
$$\cup \{(tr' - tr, ref') \mid okay \land \neg\, tr' = tr \land (A)^t_f\}$$
$$\cup \{(tr' - tr, ref') \mid okay \land \neg\, wait' \land (B)^t_f\}$$
$$\cup \{(tr' - tr, ref') \mid okay \land \neg\, tr' = tr \land (B)^t_f\}$$

[Cases on $tr' = tr$, PC, SC, ST]

$$= \{(tr' - tr, ref') \mid okay \land \neg\, wait' \land tr' = tr \land (A)^t_f\}$$ [SC and ST]
$$\cup \{(tr' - tr, ref') \mid okay \land \neg\, wait' \land \neg\, tr' = tr \land (A)^t_f\}$$
$$\cup \{(tr' - tr, ref') \mid okay \land \neg\, tr' = tr \land (A)^t_f\}$$
$$\cup \{(tr' - tr, ref') \mid okay \land \neg\, wait' \land tr' = tr \land (B)^t_f\}$$
$$\cup \{(tr' - tr, ref') \mid okay \land \neg\, wait' \land \neg\, tr' = tr \land (B)^t_f\}$$
$$\cup \{(tr' - tr, ref') \mid okay \land \neg\, tr' = tr \land (B)^t_f\}$$

$$= \{(tr' - tr, ref') \mid okay \land \neg\, wait' \land tr' = tr \land (A)^t_f\}$$ [$-$ and PC]
$$\cup \{(tr' - tr, ref') \mid okay \land \neg\, tr' = tr \land (A)^t_f\}$$
$$\cup \{(tr' - tr, ref') \mid okay \land \neg\, wait' \land tr' = tr \land (B)^t_f\}$$
$$\cup \{(tr' - tr, ref') \mid okay \land \neg\, tr' = tr \land (B)^t_f\}$$

$$= \{(\langle\rangle, ref') \mid tr' = tr \land okay \land \neg\, wait' \land (A)^t_f\}\}$$ [SC and ST]
$$\cup \{(tr' - tr, ref') \mid \neg\, tr' = tr \land okay \land (A)^t_f\}$$
$$\cup \{(\langle\rangle, ref') \mid tr' = tr \land okay \land \neg\, wait' \land (B)^t_f\}\}$$
$$\cup \{(tr' - tr, ref') \mid \neg\, tr' = tr \land okay \land (B)^t_f\}$$

$$= \{(\langle\rangle, ref') \mid tr' = tr \land okay \land \neg\, wait' \land (A)^t_f \land (B)^t_f\}$$
$$\cup \{(\langle\rangle, ref') \mid tr' = tr \land okay \land \neg\, wait' \land (A)^t_f\}\}$$
$$\cup \{(\langle\rangle, ref') \mid tr' = tr \land okay \land \neg\, wait' \land (B)^t_f\}\}$$
$$\cup \{(tr' - tr, ref') \mid \neg\, tr' = tr \land okay \land (A)^t_f\}$$
$$\cup \{(tr' - tr, ref') \mid \neg\, tr' = tr \land okay \land (B)^t_f\}$$

[Assumption and PC]

$$
\begin{aligned}
&= \{(\langle\rangle, ref') \mid tr' = tr \wedge okay \wedge (A)^t_f \wedge (B)^t_f\}\\
&\quad \cup \{(\langle\rangle, ref') \mid tr' = tr \wedge okay \wedge \neg\ wait' \wedge (A)^t_f\}\}\\
&\quad \cup \{(\langle\rangle, ref') \mid tr' = tr \wedge okay \wedge \neg\ wait' \wedge (B)^t_f\}\}\\
&\quad \cup \{(tr' - tr, ref') \mid \neg\ tr' = tr \wedge okay \wedge (A)^t_f\}\\
&\quad \cup \{(tr' - tr, ref') \mid \neg\ tr' = tr \wedge okay \wedge (B)^t_f\}
\end{aligned}
$$

[PC, SC, and ST]

$$
\begin{aligned}
&= \{(\langle\rangle, ref') \mid tr' = tr \wedge okay \wedge (A)^t_f \wedge (B)^t_f\}\\
&\quad \cup \{(\langle\rangle, ref') \mid tr' = tr \wedge okay \wedge \neg\ wait' \wedge (A)^t_f\}\}\\
&\quad \cup \{(\langle\rangle, ref') \mid tr' = tr \wedge okay \wedge \neg\ wait' \wedge (B)^t_f\}\}\\
&\quad \cup \{(tr' - tr, ref') \mid \neg\ tr' = tr \wedge okay \wedge (A)^t_f\}\\
&\quad \cup \{(tr' - tr, ref') \mid \neg\ tr' = tr \wedge okay \wedge (B)^t_f\}\\
&\quad \cup \{(\langle\rangle, ref' \cup \{\checkmark\}) \mid okay \wedge false \wedge (A)^t_f \wedge (B)^t_f\}
\end{aligned}
$$

[Assumption and PC]

$$
\begin{aligned}
&= \{(\langle\rangle, ref') \mid tr' = tr \wedge okay \wedge (A)^t_f \wedge (B)^t_f\}\\
&\quad \cup \{(\langle\rangle, ref') \mid tr' = tr \wedge okay \wedge \neg\ wait' \wedge (A)^t_f\}\}\\
&\quad \cup \{(\langle\rangle, ref') \mid tr' = tr \wedge okay \wedge \neg\ wait' \wedge (B)^t_f\}\}\\
&\quad \cup \{(tr' - tr, ref') \mid \neg\ tr' = tr \wedge okay \wedge (A)^t_f\}\\
&\quad \cup \{(tr' - tr, ref') \mid \neg\ tr' = tr \wedge okay \wedge (B)^t_f\}\\
&\quad \cup \{(\langle\rangle, ref' \cup \{\checkmark\}) \mid tr' = tr \wedge okay \wedge wait' \wedge (A)^t_f \wedge (B)^t_f\}
\end{aligned}
$$

Lemma J.25

$$
\begin{aligned}
&(okay' \wedge v' = v \wedge wait' \wedge tr' = tr \wedge (c, Sync) \notin ref');\ A^t\\
&=\\
&v' = v \wedge wait' \wedge tr' = tr \wedge (c, Sync) \notin ref'
\end{aligned}
$$

provided A is ***R3****.*

Proof.

$$
\begin{aligned}
&(okay' \wedge v' = v \wedge wait' \wedge tr' = tr \wedge (c, Sync) \notin ref');\ A^t && \text{[Proviso]}\\
&= (okay' \wedge v' = v \wedge wait' \wedge tr' = tr \wedge (c, Sync) \notin ref'); && [\mathbf{R3}]\\
&\quad (\mathbf{R3}(A))^t\\
&= (okay' \wedge v' = v \wedge wait' \wedge tr' = tr \wedge (c, Sync) \notin ref'); && \text{[PC]}\\
&\quad (II_{rea} \lhd wait \rhd A)^t\\
&= (okay' \wedge v' = v \wedge wait' \wedge tr' = tr \wedge (c, Sync) \notin ref'); && \text{[PC]}\\
&\quad (II^t_{rea} \lhd wait \rhd A^t)\\
&= (okay' \wedge v' = v \wedge wait' \wedge tr' = tr \wedge (c, Sync) \notin ref'); && [II_{rea}]\\
&\quad II^t_{rea}
\end{aligned}
$$

$$
\begin{aligned}
&= (okay' \land v' = v \land wait' \land tr' = tr \land (c, Sync) \notin ref'); && \text{[PC]}\\
&\quad \begin{pmatrix} (\neg\ okay \land tr \leq tr') \\ \lor (okay' \land tr' = tr \land wait' = wait \land ref' = ref \land v' = v) \end{pmatrix}^t \\
&= (okay' \land v' = v \land wait' \land tr' = tr \land (c, Sync) \notin ref'); && \text{[PC]}\\
&\quad (tr' = tr \land wait' = wait \land ref' = ref \land v' = v) \\
&= v' = v \land wait' \land tr' = tr \land (c, Sync) \notin ref'
\end{aligned}
$$

Lemma J.26

$$
\textbf{\textit{R1}}\left(\begin{pmatrix} ({A_1}_f^t;\ U1(out\alpha\, A_1)) \\ \land ({A_2}_f^t;\ U2(out\alpha\, A_2)) \end{pmatrix}_{+\{v,tr\}} ;\ M_{\parallel_{cs}} \right)
$$
$$
=
$$
$$
\begin{pmatrix} ({A_1}_f^t;\ U1(out\alpha\, A_1)) \\ \land ({A_2}_f^t;\ U2(out\alpha\, A_2)) \end{pmatrix}_{+\{v,tr\}} ;\ M_{\parallel_{cs}}
$$

provided

1. *A and B are* ***R***

Proof.

$$
\mathbf{R1}\left(\begin{pmatrix} ({A_1}_f^t;\ U1(out\alpha\, A_1)) \\ \land ({A_2}_f^t;\ U2(out\alpha\, A_2)) \end{pmatrix}_{+\{v,tr\}} ;\ M_{\parallel_{cs}} \right)
$$

$$\text{[}+\{v, tr\} \text{ and } M_{\parallel_{cs}}\text{]}$$

$$
= \mathbf{R1}\left(\begin{array}{l} \begin{pmatrix} ({A_1}_f^t;\ U1(out\alpha\, A_1)) \\ \land ({A_2}_f^t;\ U2(out\alpha\, A_2)) \land v' = v \land tr' = tr \end{pmatrix}; \\ \left(\begin{array}{l} tr' - tr \in (1.tr - tr \parallel_{cs} 2.tr - tr) \\ \land\ 1.tr \upharpoonright cs = 2.tr \upharpoonright cs \\ \land \left(\begin{array}{l} \begin{pmatrix} (1.wait \lor 2.wait) \\ \land\ ref' \subseteq ((1.ref \cup 2.ref) \cap cs) \cup ((1.ref \cap 2.ref) \setminus cs) \end{pmatrix} \\ \lhd wait' \rhd \\ (\neg\ 1.wait \land \neg\ 2.wait \land MSt) \end{array} \right) \end{array} \right) \end{array} \right)
$$

$$\text{[Notation } (NonTrTr')\text{]}$$

$$
= \mathbf{R1}\left(\begin{array}{l} \begin{pmatrix} ({A_1}_f^t;\ U1(out\alpha\, A_1)) \\ \land ({A_2}_f^t;\ U2(out\alpha\, A_2)) \land v' = v \land tr' = tr \end{pmatrix}; \\ \begin{pmatrix} tr' - tr \in (1.tr - tr \parallel_{cs} 2.tr - tr) \\ \land\ 1.tr \upharpoonright cs = 2.tr \upharpoonright cs \\ \land\ NonTrTr' \end{pmatrix} \end{array} \right)
$$

$$\text{[Assumption, } U1,\ U2,\ \mathbf{R1}\text{, Sequence and PC]}$$

$$= \mathbf{R1}\left(\begin{array}{l}\left(\begin{array}{l}({A_1}_f^t;\ U1(out\alpha\, A_1))\\ \land\ ({A_2}_f^t;\ U2(out\alpha\, A_2)) \land v' = v \land tr' = tr\\ \land\ tr \le 1.tr' \land tr \le 2.tr'\end{array}\right);\\ \left(\begin{array}{l}tr' - tr \in (1.tr - tr \parallel_{cs} 2.tr - tr)\\ \land\ 1.tr \upharpoonright cs = 2.tr \upharpoonright cs\\ \land\ NonTrTr'\end{array}\right)\end{array}\right)$$

[Lemma J.46]

$$= \mathbf{R1}\left(\begin{array}{l}\left(\begin{array}{l}({A_1}_f^t;\ U1(out\alpha\, A_1))\\ \land\ ({A_2}_f^t;\ U2(out\alpha\, A_2)) \land v' = v \land tr' = tr\\ \land\ tr \le 1.tr' \land tr \le 2.tr'\end{array}\right);\\ \left(\begin{array}{l}tr' \in (1.tr \parallel_{cs} 2.tr)\\ \land\ 1.tr \upharpoonright cs = 2.tr \upharpoonright cs\\ \land\ NonTrTr'\end{array}\right)\end{array}\right)$$

[Sequence, Lemma J.47 and PC]

$$= \mathbf{R1}\left(\begin{array}{l}\left(\begin{array}{l}({A_1}_f^t;\ U1(out\alpha\, A_1))\\ \land\ ({A_2}_f^t;\ U2(out\alpha\, A_2)) \land v' = v \land tr' = tr\\ \land\ tr \le 1.tr' \land tr \le 2.tr'\end{array}\right);\\ \left(\begin{array}{l}tr' \in (1.tr \parallel_{cs} 2.tr)\\ \land\ 1.tr \upharpoonright cs = 2.tr \upharpoonright cs\\ \land\ NonTrTr'\end{array}\right)\\ \land\ tr \le tr'\end{array}\right)$$

[**R1** and PC]

$$= \begin{array}{l}\left(\begin{array}{l}({A_1}_f^t;\ U1(out\alpha\, A_1))\\ \land\ ({A_2}_f^t;\ U2(out\alpha\, A_2)) \land v' = v \land tr' = tr\\ \land\ tr \le 1.tr' \land tr \le 2.tr'\end{array}\right);\\ \left(\begin{array}{l}tr' \in (1.tr \parallel_{cs} 2.tr)\\ \land\ 1.tr \upharpoonright cs = 2.tr \upharpoonright cs\\ \land\ NonTrTr'\end{array}\right)\\ \land\ tr \le tr'\end{array}$$

[Apply all steps from two above backwards]

$$=$$

$$\ldots$$

$$= \left(\begin{array}{l}({A_1}_f^t;\ U1(out\alpha\, A_1))\\ \land\ ({A_2}_f^t;\ U2(out\alpha\, A_2))\end{array}\right)_{+\{v,tr\}};\ M_{\parallel_{cs}}$$

Lemma J.27

$$\boldsymbol{R2}\left(\begin{pmatrix}(A_{1f}^{\,t};\ U1(out\alpha\, A_1))\\ \wedge\ (A_{2f}^{\,t};\ U2(out\alpha\, A_2))\end{pmatrix}_{+\{v,tr\}};\ M_{\|_{cs}}\right)$$
$$=$$
$$\begin{pmatrix}(A_{1f}^{\,t};\ U1(out\alpha\, A_1))\\ \wedge\ (A_{2f}^{\,t};\ U2(out\alpha\, A_2))\end{pmatrix}_{+\{v,tr\}};\ M_{\|_{cs}}$$

provided

1. *A and B are* $\boldsymbol{R}$

Proof.

$$\mathbf{R2}\left(\begin{pmatrix}(A_{1f}^{\,t};\ U1(out\alpha\, A_1))\\ \wedge\ (A_{2f}^{\,t};\ U2(out\alpha\, A_2))\end{pmatrix}_{+\{v,tr\}};\ M_{\|_{cs}}\right)$$

$[+\{v, tr\}$ and $M_{\|_{cs}}]$

$$= \mathbf{R2}\left(\begin{array}{l}\begin{pmatrix}(A_{1f}^{\,t};\ U1(out\alpha\, A_1))\\ \wedge\ (A_{2f}^{\,t};\ U2(out\alpha\, A_2)) \wedge v' = v \wedge tr' = tr\end{pmatrix};\\ \begin{pmatrix}tr' - tr \in (1.tr - tr \parallel_{cs} 2.tr - tr)\\ \wedge\ 1.tr \upharpoonright cs = 2.tr \upharpoonright cs\\ \wedge \begin{pmatrix}\begin{pmatrix}(1.wait \vee 2.wait)\\ \wedge\ ref' \subseteq ((1.ref \cup 2.ref) \cap cs) \cup ((1.ref \cap 2.ref) \setminus cs)\end{pmatrix}\\ \lhd wait' \rhd\\ (\neg\, 1.wait \wedge \neg\, 2.wait \wedge MSt)\end{pmatrix}\end{pmatrix}\end{array}\right)$$

[Notation $(NonTrTr')$]

$$= \mathbf{R2}\left(\begin{array}{l}\begin{pmatrix}(A_{1f}^{\,t};\ U1(out\alpha\, A_1))\\ \wedge\ (A_{2f}^{\,t};\ U2(out\alpha\, A_2)) \wedge v' = v \wedge tr' = tr\end{pmatrix};\\ \begin{pmatrix}tr' - tr \in (1.tr - tr \parallel_{cs} 2.tr - tr)\\ \wedge\ 1.tr \upharpoonright cs = 2.tr \upharpoonright cs\\ \wedge\ NonTrTr'\end{pmatrix}\end{array}\right)$$

[$\mathbf{R2}$, PC and Substitution]

$$= \begin{pmatrix}(A_{1f}^{\,t};\ U1(out\alpha\, A_1))\\ \wedge\ (A_{2f}^{\,t};\ U2(out\alpha\, A_2)) \wedge v' = v \wedge tr' = tr\end{pmatrix}[\langle\rangle/tr];$$
$$\begin{pmatrix}(tr' - tr \in (1.tr - tr \parallel_{cs} 2.tr - tr))[tr' - tr/tr']\\ \wedge\ 1.tr \upharpoonright cs = 2.tr \upharpoonright cs\\ \wedge\ NonTrTr'\end{pmatrix}$$

[Sequence, PC and Substitution]

$$= \left(\begin{array}{l} ({A_1}_f^t[\langle\rangle/tr];\ U1(out\alpha\, A_1)) \\ \land\ ({A_2}_f^t[\langle\rangle/tr];\ U2(out\alpha\, A_2)) \land v' = v \land tr' = \langle\rangle \end{array} \right);$$
$$\left(\begin{array}{l} ((tr' - tr) - tr \in (1.tr - tr \parallel_{cs} 2.tr - tr)) \\ \land\ 1.tr \upharpoonright cs = 2.tr \upharpoonright cs \\ \land\ NonTrTr' \end{array} \right)$$

[Assumption, Substitution]

$$= \left(\begin{array}{l} ({A_1}_f^t;\ U1(out\alpha\, A_1)) \\ \land\ ({A_2}_f^t;\ U2(out\alpha\, A_2)) \land v' = v \land tr' = \langle\rangle \end{array} \right);$$
$$\left(\begin{array}{l} ((tr' - tr) - tr \in (1.tr - tr \parallel_{cs} 2.tr - tr)) \\ \land\ 1.tr \upharpoonright cs = 2.tr \upharpoonright cs \\ \land\ NonTrTr' \end{array} \right)$$

[Sequence]

$$\begin{array}{l} = \exists\ okay_0, tr_0, wait_0, ref_0, v_0, \\ \quad 1.okay_0, 1.tr_0, 1.wait_0, 1.ref_0, 1.v_0, \\ \quad 2.okay_0, 2.tr_0, 2.wait_0, 2.ref_0, 2.v_0 \bullet \\ \qquad ({A_1}_f^t;\ U1(out\alpha\, A_1))[1.w_0/1.w'] \\ \qquad \land\ ({A_2}_f^t;\ U2(out\alpha\, A_2))[2.w_0/2.w'] \\ \qquad \land\ v_0 = v \land tr_0 = \langle\rangle \\ \qquad \land\ (tr' - tr_0) - tr_0 \in (1.tr_0 \parallel_{cs} 2.tr_0) \\ \qquad \land\ 1.tr_0 \upharpoonright cs = 2.tr_0 \upharpoonright cs \\ \qquad \land\ NonTrTr'[w_0, 1.w_0, 2.w_0/w, 1.w, 2.w] \end{array}$$

[PC and Sequence Property]

$$\begin{array}{l} = \exists\ okay_0, wait_0, ref_0, v_0, \\ \quad 1.okay_0, 1.tr_0, 1.wait_0, 1.ref_0, 1.v_0, \\ \quad 2.okay_0, 2.tr_0, 2.wait_0, 2.ref_0, 2.v_0 \bullet \\ \qquad ({A_1}_f^t;\ U1(out\alpha\, A_1))[1.w_0/1.w'] \\ \qquad \land\ ({A_2}_f^t;\ U2(out\alpha\, A_2))[2.w_0/2.w'] \\ \qquad \land\ v_0 = v \\ \qquad \land\ tr' \in (1.tr_0 \parallel_{cs} 2.tr_0) \\ \qquad \land\ 1.tr_0 \upharpoonright cs = 2.tr_0 \upharpoonright cs \\ \qquad \land\ NonTrTr'[w_0, 1.w_0, 2.w_0/w, 1.w, 2.w] \end{array}$$

[Lemma J.46]

$$
\begin{array}{l}
= \exists\, okay_0, wait_0, ref_0, v_0, \\
\quad 1.okay_0, 1.tr_0, 1.wait_0, 1.ref_0, 1.v_0, \\
\quad 2.okay_0, 2.tr_0, 2.wait_0, 2.ref_0, 2.v_0 \bullet \\
\qquad ({A_1}_f^t;\ U1(out\alpha\, A_1))[1.w_0/1.w'] \\
\qquad \land ({A_2}_f^t;\ U2(out\alpha\, A_2))[2.w_0/2.w'] \\
\qquad \land v_0 = v \\
\qquad \land tr' - tr \in (1.tr_0 - tr \parallel_{cs} 2.tr_0 - tr) \\
\qquad \land 1.tr_0 \upharpoonright cs = 2.tr_0 \upharpoonright cs \\
\qquad \land NonTrTr'[w_0, 1.w_0, 2.w_0 / w, 1.w, 2.w]
\end{array}
$$

[PC]

$$
\begin{array}{l}
= \exists\, okay_0, tr_0, wait_0, ref_0, v_0, \\
\quad 1.okay_0, 1.tr_0, 1.wait_0, 1.ref_0, 1.v_0, \\
\quad 2.okay_0, 2.tr_0, 2.wait_0, 2.ref_0, 2.v_0 \bullet \\
\qquad ({A_1}_f^t;\ U1(out\alpha\, A_1))[1.w_0/1.w'] \\
\qquad \land ({A_2}_f^t;\ U2(out\alpha\, A_2))[2.w_0/2.w'] \land v_0 = v \land tr_0 = tr \\
\qquad \land tr' - tr_0 \in (1.tr_0 - tr_0 \parallel_{cs} 2.tr_0 - tr_0) \\
\qquad \land 1.tr_0 \upharpoonright cs = 2.tr_0 \upharpoonright cs \\
\qquad \land NonTrTr'[w_0, 1.w_0, 2.w_0 / w, 1.w, 2.w]
\end{array}
$$

[Sequence]

$$
= \left(\begin{array}{l}
({A_1}_f^t;\ U1(out\alpha\, A_1)) \\
\land ({A_2}_f^t;\ U2(out\alpha\, A_2)) \land v' = v \land tr' = tr
\end{array}\right);
\left(\begin{array}{l}
tr' - tr \in (1.tr - tr \parallel_{cs} 2.tr - tr) \\
\land 1.tr \upharpoonright cs = 2.tr \upharpoonright cs \\
\land NonTrTr'
\end{array}\right)
$$

[Notation $NonTrTr'$]

$$
= \left(\begin{array}{l}
({A_1}_f^t;\ U1(out\alpha\, A_1)) \\
\land ({A_2}_f^t;\ U2(out\alpha\, A_2)) \land v' = v \land tr' = tr
\end{array}\right);
\left(\begin{array}{l}
tr' - tr \in (1.tr - tr \parallel_{cs} 2.tr - tr) \\
\land 1.tr \upharpoonright cs = 2.tr \upharpoonright cs \\
\land \left(\begin{array}{l}
\left(\begin{array}{l}
(1.wait \lor 2.wait) \\
\land ref' \subseteq ((1.ref \cup 2.ref) \cap cs) \cup ((1.ref \cap 2.ref) \setminus cs)
\end{array}\right) \\
\lhd wait' \rhd \\
(\neg\, 1.wait \land \neg\, 2.wait \land MSt)
\end{array}\right)
\end{array}\right)
$$

[$+\{v, tr\}$ and $M_{\parallel_{cs}}$]

$$
= \left(\begin{array}{l}
({A_1}_f^t;\ U1(out\alpha\, A_1)) \\
\land ({A_2}_f^t;\ U2(out\alpha\, A_2))
\end{array}\right)_{+\{v,tr\}};\ M_{\parallel_{cs}}
$$

Lemma J.28

$$
\begin{array}{l}
(P \llbracket ns_1 \mid cs \mid ns_2 \rrbracket Q)^t_f \\
= \\
\boldsymbol{CSP1}\left(\begin{array}{l}
\boldsymbol{R1}\left(\begin{array}{l}
\exists 1.tr', 2.tr' \bullet (A_{1}{}^f_f ;\ 1.tr' = tr) \\
\qquad \wedge (A_{2f} ;\ 2.tr' = tr) \\
\qquad \wedge 1.tr' \upharpoonright cs = 2.tr' \upharpoonright cs
\end{array}\right) \\
\vee\ \boldsymbol{R1}\left(\begin{array}{l}
\exists 1.tr', 2.tr' \bullet (A_{1f} ;\ 1.tr' = tr) \\
\qquad \wedge (A_{2}{}^f_f ;\ 2.tr' = tr) \\
\qquad \wedge 1.tr' \upharpoonright cs = 2.tr' \upharpoonright cs
\end{array}\right) \\
\vee \left(\left(\begin{array}{l}
(A_{1}{}^t_f ;\ U1(out\alpha\, A_1)) \\
\wedge (A_{2}{}^t_f ;\ U2(out\alpha\, A_2))
\end{array}\right)_{+\{v,tr\}} ;\ M_{\parallel_{cs}} \right)
\end{array}\right)
\end{array}
$$

provided

1. *A and B are* $\boldsymbol{R}$

Proof.

$$
\begin{array}{ll}
(A_1 \llbracket ns_1 \mid cs \mid ns_2 \rrbracket A_2)^t_f & [A_1 \llbracket ns_1 \mid cs \mid ns_2 \rrbracket A_2] \\
= \left(\mathbf{R} \left(\begin{array}{l}
\neg\ \exists 1.tr', 2.tr' \bullet (A_{1}{}^f_f ;\ 1.tr' = tr) \wedge (A_{2f} ;\ 2.tr' = tr) \\
\qquad \wedge 1.tr' \upharpoonright cs = 2.tr' \upharpoonright cs \\
\wedge \neg\ \exists 1.tr', 2.tr' \bullet (A_{1f} ;\ 1.tr' = tr) \wedge (A_{2}{}^f_f ;\ 2.tr' = tr) \\
\qquad \wedge 1.tr' \upharpoonright cs = 2.tr' \upharpoonright cs \\
\vdash \\
((A_{1}{}^t_f ;\ U1(out\alpha\, A_1)) \wedge (A_{2}{}^t_f ;\ U2(out\alpha\, A_2)))_{+\{v,tr\}} ;\ M_{\parallel_{cs}}
\end{array}\right)\right)^t_f & \\
& [\text{Lemma J.8}] \\
= \mathbf{CSP1}\left(\mathbf{R1}\left(\mathbf{R2}\left(\begin{array}{l}
\left(\begin{array}{l}
\neg\ \exists 1.tr', 2.tr' \bullet (A_{1}{}^f_f ;\ 1.tr' = tr) \\
\qquad \wedge (A_{2f} ;\ 2.tr' = tr) \\
\qquad \wedge 1.tr' \upharpoonright cs = 2.tr' \upharpoonright cs \\
\wedge \neg\ \exists 1.tr', 2.tr' \bullet (A_{1f} ;\ 1.tr' = tr) \\
\qquad \wedge (A_{2}{}^f_f ;\ 2.tr' = tr) \\
\qquad \wedge 1.tr' \upharpoonright cs = 2.tr' \upharpoonright cs
\end{array}\right) \Rightarrow \\
\left(\left(\begin{array}{l}
(A_{1}{}^t_f ;\ U1(out\alpha\, A_1)) \\
\wedge (A_{2}{}^t_f ;\ U2(out\alpha\, A_2))
\end{array}\right)_{+\{v,tr\}} ;\ M_{\parallel_{cs}} \right)
\end{array}\right)\right)\right) & \\
& [\text{PC}]
\end{array}
$$

$$= \mathbf{CSP1}\left(\mathbf{R1}\left(\mathbf{R2}\left(\begin{array}{l} \exists\, 1.tr', 2.tr' \bullet (A_{1}{}^{f}_{f};\ 1.tr' = tr) \\ \qquad\qquad \wedge (A_{2f};\ 2.tr' = tr) \\ \qquad\qquad \wedge\ 1.tr' \upharpoonright cs = 2.tr' \upharpoonright cs \\ \vee\ \exists\, 1.tr', 2.tr' \bullet (A_{1f};\ 1.tr' = tr) \\ \qquad\qquad \wedge (A_{2}{}^{f}_{f};\ 2.tr' = tr) \\ \qquad\qquad \wedge\ 1.tr' \upharpoonright cs = 2.tr' \upharpoonright cs \\ \vee \left(\left(\begin{array}{l} (A_{1}{}^{t}_{f};\ U1(out\alpha\, A_1)) \\ \wedge\ (A_{2}{}^{t}_{f};\ U2(out\alpha\, A_2)) \end{array} \right)_{+\{v,tr\}} ;\ M_{\|_{cs}} \right) \end{array}\right)\right)\right)$$

[**R2**, PC and Substitution]

$$= \mathbf{CSP1}\left(\mathbf{R1}\left(\begin{array}{l} \exists\, 1.tr', 2.tr' \bullet (A_{1}{}^{f}_{f};\ 1.tr' = tr) \\ \qquad\qquad \wedge (A_{2f};\ 2.tr' = tr) \\ \qquad\qquad \wedge\ 1.tr' \upharpoonright cs = 2.tr' \upharpoonright cs \\ \vee\ \exists\, 1.tr', 2.tr' \bullet (A_{1f};\ 1.tr' = tr) \\ \qquad\qquad \wedge (A_{2}{}^{f}_{f};\ 2.tr' = tr) \\ \qquad\qquad \wedge\ 1.tr' \upharpoonright cs = 2.tr' \upharpoonright cs \\ \vee\ \mathbf{R2}\left(\left(\begin{array}{l} (A_{1}{}^{t}_{f};\ U1(out\alpha\, A_1)) \\ \wedge\ (A_{2}{}^{t}_{f};\ U2(out\alpha\, A_2)) \end{array} \right)_{+\{v,tr\}} ;\ M_{\|_{cs}} \right) \end{array}\right)\right)$$

[**R1** and PC]

$$= \mathbf{CSP1}\left(\begin{array}{l} \mathbf{R1}\left(\begin{array}{l} \exists\, 1.tr', 2.tr' \bullet (A_{1}{}^{f}_{f};\ 1.tr' = tr) \\ \qquad\qquad \wedge (A_{2f};\ 2.tr' = tr) \\ \qquad\qquad \wedge\ 1.tr' \upharpoonright cs = 2.tr' \upharpoonright cs \end{array}\right) \\ \vee\ \mathbf{R1}\left(\begin{array}{l} \exists\, 1.tr', 2.tr' \bullet (A_{1f};\ 1.tr' = tr) \\ \qquad\qquad \wedge (A_{2}{}^{f}_{f};\ 2.tr' = tr) \\ \qquad\qquad \wedge\ 1.tr' \upharpoonright cs = 2.tr' \upharpoonright cs \end{array}\right) \\ \vee\ \mathbf{R1}\left(\mathbf{R2}\left(\left(\begin{array}{l} (A_{1}{}^{t}_{f};\ U1(out\alpha\, A_1)) \\ \wedge\ (A_{2}{}^{t}_{f};\ U2(out\alpha\, A_2)) \end{array} \right)_{+\{v,tr\}} ;\ M_{\|_{cs}} \right)\right) \end{array}\right)$$

[Lemma J.27 (Assumption)]

$$= \mathbf{CSP1}\left(\begin{array}{l} \mathbf{R1}\left(\begin{array}{l} \exists\, 1.tr', 2.tr' \bullet (A_{1}{}^{f}_{f};\ 1.tr' = tr) \\ \qquad\qquad \wedge (A_{2f};\ 2.tr' = tr) \\ \qquad\qquad \wedge\ 1.tr' \upharpoonright cs = 2.tr' \upharpoonright cs \end{array}\right) \\ \vee\ \mathbf{R1}\left(\begin{array}{l} \exists\, 1.tr', 2.tr' \bullet (A_{1f};\ 1.tr' = tr) \\ \qquad\qquad \wedge (A_{2}{}^{f}_{f};\ 2.tr' = tr) \\ \qquad\qquad \wedge\ 1.tr' \upharpoonright cs = 2.tr' \upharpoonright cs \end{array}\right) \\ \vee\ \mathbf{R1}\left(\left(\begin{array}{l} (A_{1}{}^{t}_{f};\ U1(out\alpha\, A_1)) \\ \wedge\ (A_{2}{}^{t}_{f};\ U2(out\alpha\, A_2)) \end{array} \right)_{+\{v,tr\}} ;\ M_{\|_{cs}} \right) \end{array}\right)$$

[Lemma J.26 (Assumption)]

$$= \mathbf{CSP1}\left(\begin{array}{l} \mathbf{R1}\left(\begin{array}{r} \exists\, 1.tr', 2.tr' \bullet ({A_1}_f^f;\ 1.tr' = tr) \\ \land\ ({A_2}_f;\ 2.tr' = tr) \\ \land\ 1.tr' \upharpoonright cs = 2.tr' \upharpoonright cs \end{array}\right) \\ \lor\ \mathbf{R1}\left(\begin{array}{r} \exists\, 1.tr', 2.tr' \bullet ({A_1}_f;\ 1.tr' = tr) \\ \land\ ({A_2}_f^f;\ 2.tr' = tr) \\ \land\ 1.tr' \upharpoonright cs = 2.tr' \upharpoonright cs \end{array}\right) \\ \lor\ \left(\left(\begin{array}{l} ({A_1}_f^t;\ U1(out\alpha\, A_1)) \\ \land\ ({A_2}_f^t;\ U2(out\alpha\, A_2)) \end{array}\right)_{+\{v,tr\}};\ M_{\parallel_{cs}}\right) \end{array}\right)$$

Lemma J.29

$$\begin{array}{l} \bigcup\{s \underset{cs}{\parallel} t \mid s \in SS \land t \in TT\} \\ = \\ \bigcup\{s \underset{cs}{\parallel} t \mid s \in SS \land t \in TT \land s \upharpoonright cs = t \upharpoonright cs\} \end{array}$$

Proof.

$$\begin{array}{lr} \bigcup\{s \underset{cs}{\parallel} t \mid s \in SS \land t \in TT\} & [\text{Notation}] \\ = \bigcup\{s, t \mid s \in SS \land t \in TT \bullet s \underset{cs}{\parallel} t\} & [\bigcup] \\ = \{x, s, t \mid s \in SS \land t \in TT \land x \in s \underset{cs}{\parallel} t \bullet x\} & [\text{ST and PC}] \\ = \{x, s, t \mid s \in SS \land t \in TT \land x \in s \underset{cs}{\parallel} t \land s \underset{cs}{\parallel} t \neq \emptyset \bullet x\} & \\ & [\text{Lemma J.52 and PC}] \\ = \{x, s, t \mid s \in SS \land t \in TT \land x \in s \underset{cs}{\parallel} t \land s \underset{cs}{\parallel} t \neq \emptyset \land s \upharpoonright cs = t \upharpoonright cs \bullet x\} & \\ & [\text{ST and PC}] \\ = \{x, s, t \mid s \in SS \land t \in TT \land s \upharpoonright cs = t \upharpoonright cs \land x \in s \underset{cs}{\parallel} t \bullet x\} & [\bigcup] \\ = \bigcup\{s, t \mid s \in SS \land t \in TT \land s \upharpoonright cs = t \upharpoonright cs \bullet s \underset{cs}{\parallel} t\} & [\text{Notation}] \\ = \bigcup\{s \underset{cs}{\parallel} t \mid s \in SS \land t \in TT \land s \upharpoonright cs = t \upharpoonright cs\} & \end{array}$$

Lemma J.30

$$\left\{\begin{array}{l} tr' - tr \mid \exists\, 1.w_0, 2.w_0 \bullet \\ \qquad P[1.w_0/w'] \wedge Q[2.w_0/w'] \\ \qquad \wedge\, tr' - tr \in (1.tr_0 - tr \parallel_{cs} 2.tr_0 - tr) \\ \qquad \wedge\, 1.tr_0 \upharpoonright cs = 2.tr_0 \upharpoonright cs \end{array}\right\}$$
$$= \bigcup \left\{\begin{array}{l} s \parallel_{cs} t \mid s \in \{tr' - tr \mid P\} \\ \qquad \wedge\, t \in \{tr' - tr \mid Q\} \\ \qquad \wedge\, s \upharpoonright cs = t \upharpoonright cs \end{array}\right\}$$

where *w contains all UTP observational variables and state components.*

Proof.

$$\bigcup \left\{\begin{array}{l} s \parallel_{cs} t \mid s \in \{tr' - tr \mid P\} \\ \qquad \wedge\, t \in \{tr' - tr \mid Q\} \\ \qquad \wedge\, s \upharpoonright cs = t \upharpoonright cs \end{array}\right\} \qquad \text{[Notation]}$$

$$= \bigcup \left\{\begin{array}{l} s, t \\ \mid s \in \{w, w' \mid P \bullet tr' - tr\} \\ \;\; \wedge\, t \in \{w, w' \mid Q \bullet tr' - tr\} \\ \;\; \wedge\, s \upharpoonright cs = t \upharpoonright cs \\ \bullet\, s \parallel_{cs} t \end{array}\right\} \qquad [\textstyle\bigcup]$$

$$= \left\{\begin{array}{l} s, t, x \\ \mid s \in \{w, w' \mid P \bullet tr' - tr\} \\ \;\; \wedge\, t \in \{w, w' \mid Q \bullet tr' - tr\} \\ \;\; \wedge\, s \upharpoonright cs = t \upharpoonright cs \\ \;\; \wedge\, x \in s \parallel_{cs} t \\ \bullet\, x \end{array}\right\} \qquad \text{[Variable Renaming]}$$

$$= \left\{\begin{array}{l} tr', s, t \\ \mid s \in \{w, w' \mid P \bullet tr' - tr\} \\ \;\; \wedge\, t \in \{w, w' \mid Q \bullet tr' - tr\} \\ \;\; \wedge\, tr' \in (s \parallel_{cs} t) \\ \;\; \wedge\, s \upharpoonright cs = t \upharpoonright cs \\ \bullet\, tr' \end{array}\right\} \qquad \text{[SC and Property of } s \parallel_{cs} t\text{]}$$

$$= \left\{\begin{array}{l} tr, tr', s, t \\ \mid s \in \{w, w' \mid P \bullet tr'\} \\ \;\; \wedge\, t \in \{w, w' \mid Q \bullet tr'\} \\ \;\; \wedge\, tr' \in (s - tr \parallel_{cs} t - tr) \\ \;\; \wedge\, s \upharpoonright cs = t \upharpoonright cs \\ \bullet\, tr' \end{array}\right\} \qquad \text{[SC and Property of } s \parallel_{cs} t\text{]}$$

$$= \left\{ \begin{array}{l} tr, tr', s, t \\ \mid s \in \{w, w' \mid P \bullet tr'\} \\ \quad \wedge\ t \in \{w, w' \mid Q \bullet tr'\} \\ \quad \wedge\ tr' \in (s \parallel_{cs} t) \\ \quad \wedge\ s \upharpoonright cs = t \upharpoonright cs \\ \bullet\ tr' - tr \end{array} \right\} \qquad \text{[Only } tr \text{ and } tr' \text{ are quantified]}$$

$$= \left\{ \begin{array}{l} w, w', s, t \\ \mid s \in \{w, w' \mid P \bullet tr'\} \\ \quad \wedge\ t \in \{w, w' \mid Q \bullet tr'\} \\ \quad \wedge\ tr' \in (s \parallel_{cs} t) \\ \quad \wedge\ s \upharpoonright cs = t \upharpoonright cs \\ \bullet\ tr' - tr \end{array} \right\} \qquad \text{[Lemma J.46]}$$

$$= \left\{ \begin{array}{l} w, w', s, t \\ \mid s \in \{w, w' \mid P \bullet tr'\} \\ \quad \wedge\ t \in \{w, w' \mid Q \bullet tr'\} \\ \quad \wedge\ tr' - tr \in (s - tr \parallel_{cs} t - tr) \\ \quad \wedge\ s \upharpoonright cs = t \upharpoonright cs \\ \bullet\ tr' - tr \end{array} \right\} \qquad \text{[PC and SC]}$$

$$= \left\{ \begin{array}{l} w, w' \\ \mid \exists\, 1.w_0, 2.w_0 \bullet \\ \qquad P[1.w_0/w'] \wedge Q[2.w_0/w'] \\ \qquad \wedge\ tr' - tr \in (1.tr_0 - tr \parallel_{cs} 2.tr_0 - tr) \\ \qquad \wedge\ 1.tr_0 \upharpoonright cs = 2.tr_0 \upharpoonright cs \\ \bullet\ tr' - tr \end{array} \right\} \qquad \text{[Notation]}$$

$$= \left\{ \begin{array}{l} tr' - tr \mid \exists\, 1.w_0, 2.w_0 \bullet \\ \qquad\qquad P[1.w_0/w'] \wedge Q[2.w_0/w'] \\ \qquad\qquad \wedge\ tr' - tr \in (1.tr_0 - tr \parallel_{cs} 2.tr_0 - tr) \\ \qquad\qquad \wedge\ 1.tr_0 \upharpoonright cs = 2.tr_0 \upharpoonright cs \end{array} \right\}$$

Lemma J.31

$$\left\{\begin{array}{l} tr' - tr \mid \\ \qquad okay \wedge \neg\, wait' \\ \qquad \wedge \left(\begin{array}{l} \left(\begin{array}{l} (P^t_f;\ U1(out\alpha\, P)) \\ \wedge\ (Q^t_f;\ U2(out\alpha\, Q)) \end{array}\right)_{+\{v,tr\}} ; \\ \left(\begin{array}{l} tr' - tr \in (1.tr - tr \parallel_{cs} 2.tr - tr) \\ \wedge\ 1.tr \upharpoonright cs = 2.tr \upharpoonright cs \\ \wedge\ (\neg\, 1.wait \wedge \neg\, 2.wait \wedge MSt) \end{array}\right) \end{array}\right) \end{array}\right\}$$

$$=$$

$$\bigcup \left\{\begin{array}{l} s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \mid s \in \{tr' - tr \mid okay \wedge \neg\, wait' \wedge (P)^t_f\} \\ \qquad \wedge\ t \in \{tr' - tr \mid okay \wedge \neg\, wait' \wedge (Q)^t_f\} \end{array}\right\}$$

Proof.

$$\left\{\begin{array}{l} tr' - tr \mid \\ \qquad okay \wedge \neg\, wait' \\ \qquad \wedge \left(\begin{array}{l} \left(\begin{array}{l} (P^t_f;\ U1(out\alpha\, P)) \\ \wedge\ (Q^t_f;\ U2(out\alpha\, Q)) \end{array}\right)_{+\{v,tr\}} ; \\ \left(\begin{array}{l} tr' - tr \in (1.tr - tr \parallel_{cs} 2.tr - tr) \\ \wedge\ 1.tr \upharpoonright cs = 2.tr \upharpoonright cs \\ \wedge\ (\neg\, 1.wait \wedge \neg\, 2.wait \wedge MSt) \end{array}\right) \end{array}\right) \end{array}\right\}$$

[Case Analysis on $wait'$ and PC]

$$\left\{\begin{array}{l} tr' - tr \mid \\ \qquad okay \\ \qquad \wedge \left(\begin{array}{l} \left(\begin{array}{l} (P^t_f;\ U1(out\alpha\, P)) \\ \wedge\ (Q^t_f;\ U2(out\alpha\, Q)) \end{array}\right)_{+\{v,tr\}} ; \\ \left(\begin{array}{l} tr' - tr \in (1.tr - tr \parallel_{cs} 2.tr - tr) \\ \wedge\ 1.tr \upharpoonright cs = 2.tr \upharpoonright cs \\ \wedge\ (\neg\, 1.wait \wedge \neg\, 2.wait \wedge MSt) \end{array}\right) \end{array}\right) \end{array}\right\}$$

[U_n, $A_{+\{v,tr\}}$ and Sequence Composition]

$$= \left\{ \begin{array}{l} tr' - tr \mid \\ \quad okay \\ \quad \wedge \left(\begin{array}{l} \left(\begin{array}{l} \left(P_f^t \left[\begin{array}{l} 1.okay', 1.wait', 1.tr', 1.ref', 1.v'/ \\ okay', wait', tr', ref', v' \end{array} \right] \right) \\ \wedge \left(Q_f^t \left[\begin{array}{l} 2.okay', 2.wait', 2.tr', 2.ref', 2.v'/ \\ okay', wait', tr', ref', v' \end{array} \right] \right) \\ v' = v \wedge tr' = tr \end{array} \right) ; \\ \left(\begin{array}{l} tr' - tr \in (1.tr - tr \parallel_{cs} 2.tr - tr) \\ \wedge 1.tr \upharpoonright cs = 2.tr \upharpoonright cs \\ \wedge (\neg 1.wait \wedge \neg 2.wait \wedge MSt) \end{array} \right) \end{array} \right) \end{array} \right\}$$

[Sequence Composition]

$$= \left\{ \begin{array}{l} tr' - tr \mid \\ \quad okay \\ \quad \wedge \left(\begin{array}{l} \exists\, okay_0, wait_0, tr_0, ref_0, v_0, \\ \quad 1.okay_0, 1.wait_0, 1.tr_0, 1.ref_0, 1.v_0, \\ \quad 2.okay_0, 2.wait_0, 2.tr_0, 2.ref_0, 2.v_0 \bullet \\ \qquad \left(P_f^t \left[\begin{array}{l} 1.okay_0, 1.wait_0, 1.tr_0, 1.ref_0, 1.v_0/ \\ okay', wait', tr', ref', v' \end{array} \right] \right) \\ \qquad \wedge \left(Q_f^t \left[\begin{array}{l} 2.okay_0, 2.wait_0, 2.tr_0, 2.ref_0, 2.v_0/ \\ okay', wait', tr', ref', v' \end{array} \right] \right) \\ \qquad v_0 = v \wedge tr_0 = tr \\ \qquad \wedge\, tr' - tr_0 \in (1.tr_0 - tr_0 \parallel_{cs} 2.tr_0 - tr_0) \\ \qquad \wedge\, 1.tr_0 \upharpoonright cs = 2.tr_0 \upharpoonright cs \\ \qquad \wedge \neg 1.wait_0 \wedge \neg 2.wait_0 \\ \qquad \wedge\, MSt[1.v_0, 2.v_0, v_0/1.v, 2.v, v] \end{array} \right) \end{array} \right\}$$

[MSt and Substitution]

$$= \left\{ \begin{array}{l} tr' - tr \mid \\ \quad okay \\ \quad \wedge \left(\begin{array}{l} \exists\, okay_0, wait_0, tr_0, ref_0, v_0, \\ \quad 1.okay_0, 1.wait_0, 1.tr_0, 1.ref_0, 1.v_0, \\ \quad 2.okay_0, 2.wait_0, 2.tr_0, 2.ref_0, 2.v_0 \bullet \\ \qquad \left(P_f^t \left[\begin{array}{l} 1.okay_0, 1.wait_0, 1.tr_0, 1.ref_0, 1.v_0 / \\ okay', wait', tr', ref', v' \end{array} \right] \right) \\ \qquad \wedge \left(Q_f^t \left[\begin{array}{l} 2.okay_0, 2.wait_0, 2.tr_0, 2.ref_0, 2.v_0 / \\ okay', wait', tr', ref', v' \end{array} \right] \right) \\ \qquad v_0 = v \wedge tr_0 = tr \\ \qquad \wedge\, tr' - tr_0 \in (1.tr_0 - tr_0 \parallel_{cs} 2.tr_0 - tr_0) \\ \qquad \wedge\, 1.tr_0 \upharpoonright cs = 2.tr_0 \upharpoonright cs \\ \qquad \wedge \neg\, 1.wait_0 \wedge \neg\, 2.wait_0 \\ \qquad \wedge \left(\begin{array}{l} \forall\, v \bullet v \in ns_1 \Rightarrow v' = 1.v_0 \\ \qquad \wedge\, v \in ns_2 \Rightarrow v' = 2.v_0 \\ \qquad \wedge\, v \notin ns_1 \cup ns_2 \Rightarrow v' = v_0 \end{array} \right) \end{array} \right) \end{array} \right\}$$

[PC]

$$= \left\{ \begin{array}{l} tr' - tr \mid \\ \quad okay \\ \quad \wedge \left(\begin{array}{l} \exists\, 1.okay_0, 1.wait_0, 1.tr_0, 1.ref_0, 1.v_0, \\ \quad 2.okay_0, 2.wait_0, 2.tr_0, 2.ref_0, 2.v_0 \bullet \\ \qquad \left(P_f^t \left[\begin{array}{l} 1.okay_0, 1.wait_0, 1.tr_0, 1.ref_0, 1.v_0 / \\ okay', wait', tr', ref', v' \end{array} \right] \right) \\ \qquad \wedge \left(Q_f^t \left[\begin{array}{l} 2.okay_0, 2.wait_0, 2.tr_0, 2.ref_0, 2.v_0 / \\ okay', wait', tr', ref', v' \end{array} \right] \right) \\ \qquad \wedge\, tr' - tr \in (1.tr_0 - tr \parallel_{cs} 2.tr_0 - tr) \\ \qquad \wedge\, 1.tr_0 \upharpoonright cs = 2.tr_0 \upharpoonright cs \\ \qquad \wedge \neg\, 1.wait_0 \wedge \neg\, 2.wait_0 \\ \qquad \wedge \left(\begin{array}{l} \forall\, v \bullet v \in ns_1 \Rightarrow v' = 1.v_0 \\ \qquad \wedge\, v \in ns_2 \Rightarrow v' = 2.v_0 \\ \qquad \wedge\, v \notin ns_1 \cup ns_2 \Rightarrow v' = v \end{array} \right) \end{array} \right) \end{array} \right\}$$

[PC]

$$= \left\{ \begin{array}{l} tr' - tr \mid \\ \quad \exists\, 1.okay_0, 1.wait_0, 1.tr_0, 1.ref_0, 1.v_0, \\ \quad\ 2.okay_0, 2.wait_0, 2.tr_0, 2.ref_0, 2.v_0 \bullet \\ \qquad \left((okay \land \neg\, wait' \land P^t_f) \left[\begin{array}{l} 1.okay_0, 1.wait_0, 1.tr_0, 1.ref_0, 1.v_0 / \\ okay', wait', tr', ref', v' \end{array} \right] \right) \\ \qquad \land \left((okay \land \neg\, wait' \land Q^t_f) \left[\begin{array}{l} 2.okay_0, 2.wait_0, 2.tr_0, 2.ref_0, 2.v_0 / \\ okay', wait', tr', ref', v' \end{array} \right] \right) \\ \qquad \land\ tr' - tr \in (1.tr_0 - tr \parallel_{cs} 2.tr_0 - tr) \\ \qquad \land\ 1.tr_0 \upharpoonright cs = 2.tr_0 \upharpoonright cs \\ \qquad \land \left(\begin{array}{l} \forall\, v \bullet v \in ns_1 \Rightarrow v' = 1.v_0 \\ \qquad \land\ v \in ns_2 \Rightarrow v' = 2.v_0 \\ \qquad \land\ v \notin ns_1 \cup ns_2 \Rightarrow v' = v \end{array} \right) \end{array} \right\}$$

[PC (v' is implicitly quantified in this notation)]

$$= \left\{ \begin{array}{l} tr' - tr \mid \\ \quad \exists\, 1.okay_0, 1.wait_0, 1.tr_0, 1.ref_0, 1.v_0, \\ \quad\ 2.okay_0, 2.wait_0, 2.tr_0, 2.ref_0, 2.v_0 \bullet \\ \qquad \left((okay \land \neg\, wait' \land P^t_f) \left[\begin{array}{l} 1.okay_0, 1.wait_0, 1.tr_0, 1.ref_0, 1.v_0 / \\ okay', wait', tr', ref', v' \end{array} \right] \right) \\ \qquad \land \left((okay \land \neg\, wait' \land Q^t_f) \left[\begin{array}{l} 2.okay_0, 2.wait_0, 2.tr_0, 2.ref_0, 2.v_0 / \\ okay', wait', tr', ref', v' \end{array} \right] \right) \\ \qquad \land\ tr' - tr \in (1.tr_0 - tr \parallel_{cs} 2.tr_0 - tr) \\ \qquad \land\ 1.tr_0 \upharpoonright cs = 2.tr_0 \upharpoonright cs \end{array} \right\}$$

[Lemma J.30]

$$= \bigcup \left\{ \begin{array}{l} s \parallel_{cs} t \mid s \in \{tr' - tr \mid okay \land \neg\, wait' \land (P)^t_f\} \\ \qquad \land\ t \in \{tr' - tr \mid okay \land \neg\, wait' \land (Q)^t_f\} \\ \qquad \land\ s \upharpoonright cs = t \upharpoonright cs \end{array} \right\}$$

[Lemma J.50]

$$= \bigcup \left\{ \begin{array}{l} s \underset{\mathsf{cs}}{\parallel} t \mid s \in \{tr' - tr \mid okay \land \neg\, wait' \land (P)^t_f\} \\ \qquad \land\ t \in \{tr' - tr \mid okay \land \neg\, wait' \land (Q)^t_f\} \\ \qquad \land\ s \upharpoonright cs = t \upharpoonright cs \end{array} \right\}$$

[Lemma J.29]

$$= \bigcup \left\{ \begin{array}{l} s \underset{\mathsf{cs}}{\parallel} t \mid s \in \{tr' - tr \mid okay \land \neg\, wait' \land (P)^t_f\} \\ \qquad \land\ t \in \{tr' - tr \mid okay \land \neg\, wait' \land (Q)^t_f\} \end{array} \right\}$$

[Lemma J.51 ($tr, tr' : \operatorname{seq} \Sigma$ and $\checkmark \notin \Sigma$)]

$$= \bigcup \left\{ \begin{array}{l} s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \mid s \in \{tr' - tr \mid okay \land \neg\, wait' \land (P)^t_f\} \\ \qquad \land\ t \in \{tr' - tr \mid okay \land \neg\, wait' \land (Q)^t_f\} \end{array} \right\}$$

Lemma J.32

$$\left\{ \begin{array}{l} (tr' - tr) \frown \langle \checkmark \rangle \mid \\ \qquad okay \land \neg\, wait' \\ \qquad \land \left(\begin{array}{l} \left(\begin{array}{l} (P^t_f;\ U1(out\alpha\, P)) \\ \land\, (Q^t_f;\ U2(out\alpha\, Q)) \end{array} \right)_{+\{v,tr\}} ; \\ \left(\begin{array}{l} tr' - tr \in (1.tr - tr \parallel_{cs} 2.tr - tr) \\ \land\, 1.tr \upharpoonright cs = 2.tr \upharpoonright cs \\ \land\, (\neg\, 1.wait \land \neg\, 2.wait \land MSt) \end{array} \right) \end{array} \right) \end{array} \right\}$$

$$=$$

$$\bigcup \left\{ \begin{array}{l} s \frown \langle \checkmark \rangle \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \frown \langle \checkmark \rangle \mid s \in \{tr' - tr \mid okay \land \neg\, wait' \land (P)^t_f\} \\ \qquad\qquad\qquad\qquad \land\, t \in \{tr' - tr \mid okay \land \neg\, wait' \land (Q)^t_f\} \end{array} \right\}$$

Proof.

$$\left\{ \begin{array}{l} (tr' - tr) \frown \langle \checkmark \rangle \mid \\ \qquad okay \land \neg\, wait' \\ \qquad \land \left(\begin{array}{l} \left(\begin{array}{l} (P^t_f;\ U1(out\alpha\, P)) \\ \land\, (Q^t_f;\ U2(out\alpha\, Q)) \end{array} \right)_{+\{v,tr\}} ; \\ \left(\begin{array}{l} tr' - tr \in (1.tr - tr \parallel_{cs} 2.tr - tr) \\ \land\, 1.tr \upharpoonright cs = 2.tr \upharpoonright cs \\ \land\, (\neg\, 1.wait \land \neg\, 2.wait \land MSt) \end{array} \right) \end{array} \right) \end{array} \right\}$$

[Case Analysis on $wait'$ and PC]

$$\left\{ \begin{array}{l} (tr' - tr) \frown \langle \checkmark \rangle \mid \\ \qquad okay \\ \qquad \land \left(\begin{array}{l} \left(\begin{array}{l} (P^t_f;\ U1(out\alpha\, P)) \\ \land\, (Q^t_f;\ U2(out\alpha\, Q)) \end{array} \right)_{+\{v,tr\}} ; \\ \left(\begin{array}{l} tr' - tr \in (1.tr - tr \parallel_{cs} 2.tr - tr) \\ \land\, 1.tr \upharpoonright cs = 2.tr \upharpoonright cs \\ \land\, (\neg\, 1.wait \land \neg\, 2.wait \land MSt) \end{array} \right) \end{array} \right) \end{array} \right\}$$

[U_n, $A_{+\{v,tr\}}$ and Sequence Composition]

$$
= \left\{ \begin{array}{l}
(tr' - tr) ^\frown \langle \checkmark \rangle \mid \\
\quad okay \\
\quad \wedge \left(\begin{array}{l}
\left(\begin{array}{l}
\left(P^t_f \left[\begin{array}{l} 1.okay', 1.wait', 1.tr', 1.ref', 1.v' / \\ okay', wait', tr', ref', v' \end{array} \right] \right) \\
\wedge \left(Q^t_f \left[\begin{array}{l} 2.okay', 2.wait', 2.tr', 2.ref', 2.v' / \\ okay', wait', tr', ref', v' \end{array} \right] \right) \\
v' = v \wedge tr' = tr
\end{array} \right) ; \\
\left(\begin{array}{l}
tr' - tr \in (1.tr - tr \parallel_{cs} 2.tr - tr) \\
\wedge\ 1.tr \upharpoonright cs = 2.tr \upharpoonright cs \\
\wedge\ (\neg\ 1.wait \wedge \neg\ 2.wait \wedge MSt)
\end{array} \right)
\end{array} \right)
\end{array} \right\}
$$

[Sequence Composition]

$$
= \left\{ \begin{array}{l}
(tr' - tr) ^\frown \langle \checkmark \rangle \mid \\
\quad okay \\
\quad \wedge \left(\begin{array}{l}
\exists\ okay_0, wait_0, tr_0, ref_0, v_0, \\
\quad 1.okay_0, 1.wait_0, 1.tr_0, 1.ref_0, 1.v_0, \\
\quad 2.okay_0, 2.wait_0, 2.tr_0, 2.ref_0, 2.v_0 \bullet \\
\qquad \left(P^t_f \left[\begin{array}{l} 1.okay_0, 1.wait_0, 1.tr_0, 1.ref_0, 1.v_0 / \\ okay', wait', tr', ref', v' \end{array} \right] \right) \\
\qquad \wedge \left(Q^t_f \left[\begin{array}{l} 2.okay_0, 2.wait_0, 2.tr_0, 2.ref_0, 2.v_0 / \\ okay', wait', tr', ref', v' \end{array} \right] \right) \\
\qquad v_0 = v \wedge tr_0 = tr \\
\qquad \wedge\ tr' - tr_0 \in (1.tr_0 - tr_0 \parallel_{cs} 2.tr_0 - tr_0) \\
\qquad \wedge\ 1.tr_0 \upharpoonright cs = 2.tr_0 \upharpoonright cs \\
\qquad \wedge \neg\ 1.wait_0 \wedge \neg\ 2.wait_0 \\
\qquad \wedge\ MSt[1.v_0, 2.v_0, v_0 / 1.v, 2.v, v]
\end{array} \right)
\end{array} \right\}
$$

[MSt and Substitution]

$$= \left\{ \begin{array}{l} (tr' - tr) \frown \langle \checkmark \rangle \mid \\ \qquad okay \\ \qquad \land \left(\begin{array}{l} \exists\, okay_0, wait_0, tr_0, ref_0, v_0, \\ \quad 1.okay_0, 1.wait_0, 1.tr_0, 1.ref_0, 1.v_0, \\ \quad 2.okay_0, 2.wait_0, 2.tr_0, 2.ref_0, 2.v_0 \bullet \\ \qquad \left(P^t_f \left[\begin{array}{l} 1.okay_0, 1.wait_0, 1.tr_0, 1.ref_0, 1.v_0 / \\ okay', wait', tr', ref', v' \end{array} \right] \right) \\ \qquad \land \left(Q^t_f \left[\begin{array}{l} 2.okay_0, 2.wait_0, 2.tr_0, 2.ref_0, 2.v_0 / \\ okay', wait', tr', ref', v' \end{array} \right] \right) \\ \qquad v_0 = v \land tr_0 = tr \\ \qquad \land\, tr' - tr_0 \in (1.tr_0 - tr_0 \parallel_{cs} 2.tr_0 - tr_0) \\ \qquad \land\, 1.tr_0 \upharpoonright cs = 2.tr_0 \upharpoonright cs \\ \qquad \land \neg\, 1.wait_0 \land \neg\, 2.wait_0 \\ \qquad \land \left(\begin{array}{l} \forall\, v \bullet v \in ns_1 \Rightarrow v' = 1.v_0 \\ \qquad \land\, v \in ns_2 \Rightarrow v' = 2.v_0 \\ \qquad \land\, v \notin ns_1 \cup ns_2 \Rightarrow v' = v_0 \end{array} \right) \end{array} \right) \end{array} \right\}$$

[PC]

$$= \left\{ \begin{array}{l} (tr' - tr) \frown \langle \checkmark \rangle \mid \\ \qquad okay \\ \qquad \land \left(\begin{array}{l} \exists\, 1.okay_0, 1.wait_0, 1.tr_0, 1.ref_0, 1.v_0, \\ \quad 2.okay_0, 2.wait_0, 2.tr_0, 2.ref_0, 2.v_0 \bullet \\ \qquad \left(P^t_f \left[\begin{array}{l} 1.okay_0, 1.wait_0, 1.tr_0, 1.ref_0, 1.v_0 / \\ okay', wait', tr', ref', v' \end{array} \right] \right) \\ \qquad \land \left(Q^t_f \left[\begin{array}{l} 2.okay_0, 2.wait_0, 2.tr_0, 2.ref_0, 2.v_0 / \\ okay', wait', tr', ref', v' \end{array} \right] \right) \\ \qquad \land\, tr' - tr \in (1.tr_0 - tr \parallel_{cs} 2.tr_0 - tr) \\ \qquad \land\, 1.tr_0 \upharpoonright cs = 2.tr_0 \upharpoonright cs \\ \qquad \land \neg\, 1.wait_0 \land \neg\, 2.wait_0 \\ \qquad \land \left(\begin{array}{l} \forall\, v \bullet v \in ns_1 \Rightarrow v' = 1.v_0 \\ \qquad \land\, v \in ns_2 \Rightarrow v' = 2.v_0 \\ \qquad \land\, v \notin ns_1 \cup ns_2 \Rightarrow v' = v \end{array} \right) \end{array} \right) \end{array} \right\}$$

[PC (v' is implicitly quantified in this notation)]

$$= \left\{ \begin{array}{l} (tr' - tr) \frown \langle \checkmark \rangle \mid \\ \quad okay \\ \quad \wedge \left(\begin{array}{l} \exists\, 1.okay_0, 1.wait_0, 1.tr_0, 1.ref_0, 1.v_0, \\ \quad 2.okay_0, 2.wait_0, 2.tr_0, 2.ref_0, 2.v_0 \bullet \\ \quad\quad \left(P^t_f \left[\begin{array}{l} 1.okay_0, 1.wait_0, 1.tr_0, 1.ref_0, 1.v_0 / \\ okay', wait', tr', ref', v' \end{array} \right] \right) \\ \quad\quad \wedge \left(Q^t_f \left[\begin{array}{l} 2.okay_0, 2.wait_0, 2.tr_0, 2.ref_0, 2.v_0 / \\ okay', wait', tr', ref', v' \end{array} \right] \right) \\ \quad\quad \wedge\, tr' - tr \in (1.tr_0 - tr \parallel_{cs} 2.tr_0 - tr) \\ \quad\quad \wedge\, 1.tr_0 \upharpoonright cs = 2.tr_0 \upharpoonright cs \\ \quad\quad \wedge \neg\, 1.wait_0 \wedge \neg\, 2.wait_0 \end{array} \right) \end{array} \right\}$$

[PC]

$$= \left\{ \begin{array}{l} (tr' - tr) \frown \langle \checkmark \rangle \mid \exists\, 1.okay_0, 1.wait_0, 1.tr_0, 1.ref_0, 1.v_0, \\ \quad\quad 2.okay_0, 2.wait_0, 2.tr_0, 2.ref_0, 2.v_0 \bullet \\ \quad\quad\quad \left(\begin{array}{l} (okay \wedge \neg\, wait' \wedge P^t_f) \\ \quad \left[\begin{array}{l} 1.okay_0, 1.wait_0, 1.tr_0, 1.ref_0, 1.v_0 / \\ okay', wait', tr', ref', v' \end{array} \right] \end{array} \right) \\ \quad\quad\quad \wedge \left(\begin{array}{l} (okay \wedge \neg\, wait' \wedge Q^t_f) \\ \quad \left[\begin{array}{l} 2.okay_0, 2.wait_0, 2.tr_0, 2.ref_0, 2.v_0 / \\ okay', wait', tr', ref', v' \end{array} \right] \end{array} \right) \\ \quad\quad\quad \wedge\, tr' - tr \in (1.tr_0 - tr \parallel_{cs} 2.tr_0 - tr) \\ \quad\quad\quad \wedge\, 1.tr_0 \upharpoonright cs = 2.tr_0 \upharpoonright cs \end{array} \right\}$$

[SC]

$$= \left\{ \begin{array}{l} x \frown \langle \checkmark \rangle \mid \\ \quad x \in \left\{ \begin{array}{l} (tr' - tr) \mid \exists\, 1.okay_0, 1.wait_0, 1.tr_0, 1.ref_0, 1.v_0, \\ \quad\quad 2.okay_0, 2.wait_0, 2.tr_0, 2.ref_0, 2.v_0 \bullet \\ \quad\quad\quad \left(\begin{array}{l} (okay \wedge \neg\, wait' \wedge P^t_f) \\ \quad \left[\begin{array}{l} 1.okay_0, 1.wait_0, 1.tr_0, 1.ref_0, 1.v_0 / \\ okay', wait', tr', ref', v' \end{array} \right] \end{array} \right) \\ \quad\quad\quad \wedge \left(\begin{array}{l} (okay \wedge \neg\, wait' \wedge Q^t_f) \\ \quad \left[\begin{array}{l} 2.okay_0, 2.wait_0, 2.tr_0, 2.ref_0, 2.v_0 / \\ okay', wait', tr', ref', v' \end{array} \right] \end{array} \right) \\ \quad\quad\quad \wedge\, tr' - tr \in (1.tr_0 - tr \parallel_{cs} 2.tr_0 - tr) \\ \quad\quad\quad \wedge\, 1.tr_0 \upharpoonright cs = 2.tr_0 \upharpoonright cs \end{array} \right\} \end{array} \right\}$$

[Lemma J.30]

$$= \left\{ \begin{array}{l} x \frown \langle \checkmark \rangle \mid \\ \quad x \in \bigcup \left\{ \begin{array}{l} s \parallel_{cs} t \mid s \in \{(tr' - tr) \mid okay \wedge \neg\, wait' \wedge (P)^t_f\} \\ \quad\quad \wedge\, t \in \{(tr' - tr) \mid okay \wedge \neg\, wait' \wedge (Q)^t_f\} \\ \quad\quad \wedge\, s \upharpoonright cs = t \upharpoonright cs \end{array} \right\} \end{array} \right\}$$

[Lemma J.50]

$$= \left\{ \begin{array}{l} x \frown \langle \checkmark \rangle \mid \\ \qquad x \in \bigcup \left\{ \begin{array}{l} s \parallel_{cs} t \mid s \in \{(tr' - tr) \mid okay \land \neg\, wait' \land (P)^t_f\} \\ \qquad \land\ t \in \{(tr' - tr) \mid okay \land \neg\, wait' \land (Q)^t_f\} \\ \qquad \land\ s \upharpoonright cs = t \upharpoonright cs \end{array} \right\} \end{array} \right\}$$

[Lemma J.29]

$$= \left\{ \begin{array}{l} x \frown \langle \checkmark \rangle \mid \\ \qquad x \in \bigcup \left\{ \begin{array}{l} s \parallel_{cs} t \mid s \in \{(tr' - tr) \mid okay \land \neg\, wait' \land (P)^t_f\} \\ \qquad \land\ t \in \{(tr' - tr) \mid okay \land \neg\, wait' \land (Q)^t_f\} \end{array} \right\} \end{array} \right\}$$

[Lemma J.49]

$$= \left\{ x \mid x \in \bigcup \left\{ \begin{array}{l} s \frown \langle \checkmark \rangle \parallel_{cs^{\checkmark}} t \frown \langle \checkmark \rangle \mid s \in \{(tr' - tr) \mid okay \land \neg\, wait' \land (P)^t_f\} \\ \qquad\qquad \land\ t \in \{(tr' - tr) \mid okay \land \neg\, wait' \land (Q)^t_f\} \end{array} \right\} \right\}$$

[$\bigcup$ and SC]

$$= \bigcup \left\{ \begin{array}{l} s \frown \langle \checkmark \rangle \parallel_{cs^{\checkmark}} t \frown \langle \checkmark \rangle \mid s \in \{(tr' - tr) \mid okay \land \neg\, wait' \land (P)^t_f\} \\ \qquad\qquad \land\ t \in \{(tr' - tr) \mid okay \land \neg\, wait' \land (Q)^t_f\} \end{array} \right\}$$

Lemma J.33

$$\left\{ \begin{array}{l} tr' - tr \mid \\ \quad okay \land wait' \\ \quad \land \left(\begin{array}{l} \left(\begin{array}{l} (P^t_f;\ U1(out\alpha\, P)) \\ \land\ (Q^t_f;\ U2(out\alpha\, Q)) \end{array} \right)_{+\{v,tr\}} ; \\ \left(\begin{array}{l} tr' - tr \in (1.tr - tr \parallel_{cs} 2.tr - tr) \\ \land\ 1.tr \upharpoonright cs = 2.tr \upharpoonright cs \\ \land\ (1.wait \land 2.wait) \\ \land\ ref' \subseteq \left(\begin{array}{l} ((1.ref \cup 2.ref) \cap cs) \\ \cup((1.ref \cap 2.ref) \setminus cs) \end{array} \right) \end{array} \right) \end{array} \right) \end{array} \right\}$$

$$= \bigcup \left\{ \begin{array}{l} s \parallel_{cs^{\checkmark}} t \mid s \in \{tr' - tr \mid okay \land wait' \land (P)^t_f\} \\ \qquad \land\ t \in \{tr' - tr \mid okay \land wait' \land (Q)^t_f\} \end{array} \right\}$$

Proof.

$$\left\{ \begin{array}{l} tr' - tr \mid \\ \qquad okay \wedge \neg\, wait' \\ \qquad \wedge \left(\begin{array}{l} \left(\begin{array}{l} (P^t_f;\ U1(out\alpha\, P)) \\ \wedge\, (Q^t_f;\ U2(out\alpha\, Q)) \end{array} \right)_{+\{v,tr\}} ; \\ \left(\begin{array}{l} tr' - tr \in (1.tr - tr \parallel_{cs} 2.tr - tr) \\ \wedge\, 1.tr \upharpoonright cs = 2.tr \upharpoonright cs \\ \wedge\, (1.wait \wedge 2.wait) \\ \wedge\, ref' \subseteq \left(\begin{array}{l} ((1.ref \cup 2.ref) \cap cs) \\ \cup((1.ref \cap 2.ref) \setminus cs) \end{array} \right) \end{array} \right) \end{array} \right) \end{array} \right\}$$

[Case Analysis on $wait'$ and PC]

$$\left\{ \begin{array}{l} tr' - tr \mid \\ \qquad okay \\ \qquad \wedge \left(\begin{array}{l} \left(\begin{array}{l} (P^t_f;\ U1(out\alpha\, P)) \\ \wedge\, (Q^t_f;\ U2(out\alpha\, Q)) \end{array} \right)_{+\{v,tr\}} ; \\ \left(\begin{array}{l} tr' - tr \in (1.tr - tr \parallel_{cs} 2.tr - tr) \\ \wedge\, 1.tr \upharpoonright cs = 2.tr \upharpoonright cs \\ \wedge\, (1.wait \wedge 2.wait) \\ \wedge\, ref' \subseteq \left(\begin{array}{l} ((1.ref \cup 2.ref) \cap cs) \\ \cup((1.ref \cap 2.ref) \setminus cs) \end{array} \right) \end{array} \right) \end{array} \right) \end{array} \right\}$$

[U_n, $A_{+\{v,tr\}}$ and Sequence Composition]

$$= \left\{ \begin{array}{l} tr' - tr \mid \\ \qquad okay \\ \qquad \wedge \left(\begin{array}{l} \left(\begin{array}{l} \left(P^t_f \left[\begin{array}{l} 1.okay', 1.wait', 1.tr', 1.ref', 1.v' / \\ okay', wait', tr', ref', v' \end{array} \right] \right) \\ \wedge \left(Q^t_f \left[\begin{array}{l} 2.okay', 2.wait', 2.tr', 2.ref', 2.v' / \\ okay', wait', tr', ref', v' \end{array} \right] \right) \\ v' = v \wedge tr' = tr \end{array} \right) ; \\ \left(\begin{array}{l} tr' - tr \in (1.tr - tr \parallel_{cs} 2.tr - tr) \\ \wedge\, 1.tr \upharpoonright cs = 2.tr \upharpoonright cs \\ \wedge\, (1.wait \wedge 2.wait) \\ \wedge\, ref' \subseteq \left(\begin{array}{l} ((1.ref \cup 2.ref) \cap cs) \\ \cup((1.ref \cap 2.ref) \setminus cs) \end{array} \right) \end{array} \right) \end{array} \right) \end{array} \right\}$$

[Sequence Composition]

$$= \left\{ \begin{array}{l} tr' - tr \mid \\ \quad okay \\ \quad \wedge \left(\begin{array}{l} \exists\, okay_0, wait_0, tr_0, ref_0, v_0, \\ \quad 1.okay_0, 1.wait_0, 1.tr_0, 1.ref_0, 1.v_0, \\ \quad 2.okay_0, 2.wait_0, 2.tr_0, 2.ref_0, 2.v_0 \bullet \\ \qquad \left(P^t_f \left[\begin{array}{l} 1.okay_0, 1.wait_0, 1.tr_0, 1.ref_0, 1.v_0 / \\ okay', wait', tr', ref', v' \end{array} \right] \right) \\ \qquad \wedge \left(Q^t_f \left[\begin{array}{l} 2.okay_0, 2.wait_0, 2.tr_0, 2.ref_0, 2.v_0 / \\ okay', wait', tr', ref', v' \end{array} \right] \right) \\ \qquad \wedge v_0 = v \wedge tr_0 = tr \\ \qquad \wedge tr' - tr_0 \in (1.tr_0 - tr_0 \parallel_{cs} 2.tr_0 - tr_0) \\ \qquad \wedge 1.tr_0 \upharpoonright cs = 2.tr_0 \upharpoonright cs \\ \qquad \wedge (1.wait_0 \wedge 2.wait_0) \\ \qquad \wedge ref' \subseteq \left(\begin{array}{l} ((1.ref_0 \cup 2.ref_0) \cap cs) \\ \cup((1.ref_0 \cap 2.ref_0) \setminus cs) \end{array} \right) \end{array} \right) \end{array} \right\}$$

[PC]

$$= \left\{ \begin{array}{l} tr' - tr \mid \\ \quad okay \\ \quad \wedge \left(\begin{array}{l} \exists\, 1.okay_0, 1.wait_0, 1.tr_0, 1.ref_0, 1.v_0, \\ \quad 2.okay_0, 2.wait_0, 2.tr_0, 2.ref_0, 2.v_0 \bullet \\ \qquad \left(P^t_f \left[\begin{array}{l} 1.okay_0, 1.wait_0, 1.tr_0, 1.ref_0, 1.v_0 / \\ okay', wait', tr', ref', v' \end{array} \right] \right) \\ \qquad \wedge \left(Q^t_f \left[\begin{array}{l} 2.okay_0, 2.wait_0, 2.tr_0, 2.ref_0, 2.v_0 / \\ okay', wait', tr', ref', v' \end{array} \right] \right) \\ \qquad \wedge tr' - tr \in (1.tr_0 - tr \parallel_{cs} 2.tr_0 - tr) \\ \qquad \wedge 1.tr_0 \upharpoonright cs = 2.tr_0 \upharpoonright cs \\ \qquad \wedge (1.wait_0 \wedge 2.wait_0) \\ \qquad \wedge ref' \subseteq \left(\begin{array}{l} ((1.ref_0 \cup 2.ref_0) \cap cs) \\ \cup((1.ref_0 \cap 2.ref_0) \setminus cs) \end{array} \right) \end{array} \right) \end{array} \right\}$$

[PC]

$$= \left\{ \begin{array}{l} tr' - tr \mid \\ \quad \exists\, 1.okay_0, 1.wait_0, 1.tr_0, 1.ref_0, 1.v_0, \\ \quad\; 2.okay_0, 2.wait_0, 2.tr_0, 2.ref_0, 2.v_0 \bullet \\ \qquad \left((okay \wedge wait' \wedge P^t_f) \left[\begin{array}{l} 1.okay_0, 1.wait_0, 1.tr_0, 1.ref_0, 1.v_0 / \\ okay', wait', tr', ref', v' \end{array} \right] \right) \\ \qquad \wedge \left((okay \wedge wait' \wedge Q^t_f) \left[\begin{array}{l} 2.okay_0, 2.wait_0, 2.tr_0, 2.ref_0, 2.v_0 / \\ okay', wait', tr', ref', v' \end{array} \right] \right) \\ \qquad \wedge\, tr' - tr \in (1.tr_0 - tr \parallel_{cs} 2.tr_0 - tr) \\ \qquad \wedge\, 1.tr_0 \upharpoonright cs = 2.tr_0 \upharpoonright cs \\ \qquad \wedge\, ref' \subseteq \left(\begin{array}{l} ((1.ref_0 \cup 2.ref_0) \cap cs) \\ \cup((1.ref_0 \cap 2.ref_0) \setminus cs) \end{array} \right) \end{array} \right\}$$

[PC (ref' is implicitly quantified in this notation)]

$$= \left\{ \begin{array}{l} tr' - tr \mid \exists\, 1.okay_0, 1.wait_0, 1.tr_0, 1.ref_0, 1.v_0, \\ \qquad\quad 2.okay_0, 2.wait_0, 2.tr_0, 2.ref_0, 2.v_0 \bullet \\ \qquad\qquad \left((okay \wedge wait' \wedge P^t_f) \left[\begin{array}{l} 1.okay_0, 1.wait_0, 1.tr_0, 1.ref_0, 1.v_0 / \\ okay', wait', tr', ref', v' \end{array} \right] \right) \\ \qquad\qquad \wedge \left((okay \wedge wait' \wedge Q^t_f) \left[\begin{array}{l} 2.okay_0, 2.wait_0, 2.tr_0, 2.ref_0, 2.v_0 / \\ okay', wait', tr', ref', v' \end{array} \right] \right) \\ \qquad\qquad \wedge\, tr' - tr \in (1.tr_0 - tr \parallel_{cs} 2.tr_0 - tr) \\ \qquad\qquad \wedge\, 1.tr_0 \upharpoonright cs = 2.tr_0 \upharpoonright cs \end{array} \right\}$$

[Lemma J.30]

$$= \bigcup \left\{ \begin{array}{l} s \parallel_{cs} t \mid s \in \{tr' - tr \mid okay \wedge wait' \wedge (P)^t_f\} \\ \qquad\qquad \wedge\, t \in \{tr' - tr \mid okay \wedge wait' \wedge (Q)^t_f\} \\ \qquad\qquad \wedge\, s \upharpoonright cs = t \upharpoonright cs \end{array} \right\}$$

[Lemma J.50]

$$= \bigcup \left\{ \begin{array}{l} s \underset{\mathsf{cs}}{\parallel} t \mid s \in \{tr' - tr \mid okay \wedge wait' \wedge (P)^t_f\} \\ \qquad\quad \wedge\, t \in \{tr' - tr \mid okay \wedge wait' \wedge (Q)^t_f\} \\ \qquad\quad \wedge\, s \upharpoonright cs = t \upharpoonright cs \end{array} \right\}$$

[Lemma J.29]

$$= \bigcup \left\{ \begin{array}{l} s \underset{\mathsf{cs}}{\parallel} t \mid s \in \{tr' - tr \mid okay \wedge wait' \wedge (P)^t_f\} \\ \qquad\quad \wedge\, t \in \{tr' - tr \mid okay \wedge wait' \wedge (Q)^t_f\} \end{array} \right\}$$

[Lemma J.51 ($tr, tr' : \mathrm{seq}\,\Sigma$ and $\checkmark \notin \Sigma$)]

$$= \bigcup \left\{ \begin{array}{l} s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \mid s \in \{tr' - tr \mid okay \wedge wait' \wedge (P)^t_f\} \\ \qquad\quad \wedge\, t \in \{tr' - tr \mid okay \wedge wait' \wedge (Q)^t_f\} \end{array} \right\}$$

Lemma J.34

$$
\left\{ \begin{array}{l}
tr' - tr \mid \\
\qquad okay \land wait' \\
\qquad \land \left(\begin{array}{l}
\left(\begin{array}{l} (P^t_f;\ U1(out\alpha\, P)) \\ \land\ (Q^t_f;\ U2(out\alpha\, Q)) \end{array} \right)_{+\{v,tr\}} ; \\
\left(\begin{array}{l}
tr' - tr \in (1.tr - tr \parallel_{cs} 2.tr - tr) \\
\land\ 1.tr \upharpoonright cs = 2.tr \upharpoonright cs \\
\land\ (1.wait \land \neg\, 2.wait) \\
\land\ ref' \subseteq \left(\begin{array}{l} ((1.ref \cup 2.ref) \cap cs) \\ \cup((1.ref \cap 2.ref) \setminus cs) \end{array} \right)
\end{array} \right)
\end{array} \right)
\end{array} \right\}
$$

$$
= \bigcup \left\{ \begin{array}{l}
s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \mid s \in \{tr' - tr \mid okay \land wait' \land (P)^t_f\} \\
\qquad \land\ t \in \{tr' - tr \mid okay \land \neg\, wait' \land (Q)^t_f\}
\end{array} \right\}
$$

Proof. Very similar to that of Lemma J.33

Lemma J.35

$$
\left\{ \begin{array}{l}
tr' - tr \mid \\
\qquad okay \land wait' \\
\qquad \land \left(\begin{array}{l}
\left(\begin{array}{l} (P^t_f;\ U1(out\alpha\, P)) \\ \land\ (Q^t_f;\ U2(out\alpha\, Q)) \end{array} \right)_{+\{v,tr\}} ; \\
\left(\begin{array}{l}
tr' - tr \in (1.tr - tr \parallel_{cs} 2.tr - tr) \\
\land\ 1.tr \upharpoonright cs = 2.tr \upharpoonright cs \\
\land\ (\neg\, 1.wait \land 2.wait) \\
\land\ ref' \subseteq \left(\begin{array}{l} ((1.ref \cup 2.ref) \cap cs) \\ \cup((1.ref \cap 2.ref) \setminus cs) \end{array} \right)
\end{array} \right)
\end{array} \right)
\end{array} \right\}
$$

$$
= \bigcup \left\{ \begin{array}{l}
s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \mid s \in \{tr' - tr \mid okay \land \neg\, wait' \land (P)^t_f\} \\
\qquad \land\ t \in \{tr' - tr \mid okay \land wait' \land (Q)^t_f\}
\end{array} \right\}
$$

Proof. Very similar to that of Lemma J.33

Lemma J.36

$$\left\{\begin{array}{l} (tr' - tr, ref') \mid \exists\, 1.w_0, 2.w_0 \bullet \\ \qquad P[1.w_0/w'] \wedge Q[2.w_0/w'] \\ \qquad \wedge\ tr' - tr \in (1.tr_0 - tr \parallel_{cs} 2.tr_0 - tr) \\ \qquad \wedge\ 1.tr_0 \upharpoonright cs = 2.tr_0 \upharpoonright cs \\ \qquad \wedge\ ref' \subseteq \left(\begin{array}{l} ((1.ref_0 \cup 2.ref_0) \cap cs) \\ \cup((1.ref_0 \cap 2.ref_0) \setminus cs) \end{array}\right) \end{array}\right\}$$

$$= \left\{\begin{array}{l} (u, Y \cup Z) \\ \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \quad \wedge\ \exists\, s, t \bullet (s, Y) \in \{(tr' - tr, ref') \mid P\} \\ \qquad \wedge\ (t, Z) \in \{(tr' - tr, ref') \mid Q\} \\ \qquad \wedge\ u \in s \parallel_{cs} t \\ \qquad \wedge\ s \upharpoonright cs = t \upharpoonright cs \end{array}\right\}$$

where *w contains all UTP observational variables and state components.*

Proof.

$$\left\{\begin{array}{l} (u, Y \cup Z) \\ \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \quad \wedge\ \exists\, s, t \bullet (s, Y) \in \{(tr' - tr, ref') \mid P\} \\ \qquad \wedge\ (t, Z) \in \{(tr' - tr, ref') \mid Q\} \\ \qquad \wedge\ u \in s \parallel_{cs} t \\ \qquad \wedge\ s \upharpoonright cs = t \upharpoonright cs \end{array}\right\} \qquad [\text{Notation}]$$

$$= \left\{\begin{array}{l} u, Y, Z \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \qquad \wedge\ \exists\, s, t \bullet (s, Y) \in \{w, w' \mid P \bullet (tr' - tr, ref')\} \\ \qquad\qquad \wedge\ (t, Z) \in \{w, w' \mid Q \bullet (tr' - tr, ref')\} \\ \qquad\qquad \wedge\ u \in s \parallel_{cs} t \\ \qquad\qquad \wedge\ s \upharpoonright cs = t \upharpoonright cs \\ \bullet\ (u, Y \cup Z) \end{array}\right\}$$

[Variable Renaming]

$$= \left\{\begin{array}{l} tr', Y, Z \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \qquad \wedge\ \exists\, s, t \bullet (s, Y) \in \{w, w' \mid P \bullet (tr' - tr, ref')\} \\ \qquad\qquad \wedge\ (t, Z) \in \{w, w' \mid Q \bullet (tr' - tr, ref')\} \\ \qquad\qquad \wedge\ tr' \in s \parallel_{cs} t \\ \qquad\qquad \wedge\ s \upharpoonright cs = t \upharpoonright cs \\ \bullet\ (tr', Y \cup Z) \end{array}\right\}$$

[SC and Property of $s \parallel_{cs} t$]

$$= \left\{ \begin{array}{l} tr, tr', Y, Z \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \qquad \land \exists\, s, t \bullet (s, Y) \in \{w, w' \mid P \bullet (tr', ref')\} \\ \qquad\qquad \land (t, Z) \in \{w, w' \mid Q \bullet (tr', ref')\} \\ \qquad\qquad \land tr' \in s - tr \parallel_{cs} t - tr \\ \qquad\qquad \land s \upharpoonright cs = t \upharpoonright cs \\ \bullet (tr', Y \cup Z) \end{array} \right\}$$

[SC and Property of $s \parallel_{cs} t$]

$$\left\{ \begin{array}{l} tr, tr', Y, Z \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \qquad \land \exists\, s, t \bullet (s, Y) \in \{w, w' \mid P \bullet (tr', ref')\} \\ \qquad\qquad \land (t, Z) \in \{w, w' \mid Q \bullet (tr', ref')\} \\ \qquad\qquad \land tr' \in s \parallel_{cs} t \\ \qquad\qquad \land s \upharpoonright cs = t \upharpoonright cs \\ \bullet (tr' - tr, Y \cup Z) \end{array} \right\}$$

[Only tr and tr' are quantified]

$$= \left\{ \begin{array}{l} w, w', Y, Z \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \qquad \land \exists\, s, t \bullet (s, Y) \in \{w, w' \mid P \bullet (tr', ref')\} \\ \qquad\qquad \land (t, Z) \in \{w, w' \mid Q \bullet (tr', ref')\} \\ \qquad\qquad \land tr' \in s \parallel_{cs} t \\ \qquad\qquad \land s \upharpoonright cs = t \upharpoonright cs \\ \qquad\qquad \land ref' = Y \cup Z \\ \bullet (tr' - tr, ref') \end{array} \right\}$$

[Lemma J.46]

$$= \left\{ \begin{array}{l} w, w', Y, Z \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \qquad \land \exists\, s, t \bullet (s, Y) \in \{w, w' \mid P \bullet (tr', ref')\} \\ \qquad\qquad \land (t, Z) \in \{w, w' \mid Q \bullet (tr', ref')\} \\ \qquad\qquad \land tr' - tr \in (s - tr \parallel_{cs} t - tr) \\ \qquad\qquad \land s \upharpoonright cs = t \upharpoonright cs \\ \qquad\qquad \land ref' = Y \cup Z \\ \bullet (tr' - tr, ref') \end{array} \right\}$$

[PC and SC]

$$= \left\{ \begin{array}{l} w, w', Y, Z, s, t \mid (s, Y) \in \{w, w' \mid P \bullet (tr', ref')\} \\ \qquad \land (t, Z) \in \{w, w' \mid Q \bullet (tr', ref')\} \\ \qquad \land tr' - tr \in (s - tr \parallel_{cs} t - tr) \\ \qquad \land s \upharpoonright cs = t \upharpoonright cs \\ \qquad \land ref' = Y \cup Z \\ \qquad \land Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \bullet (tr' - tr, ref') \end{array} \right\}$$

[ST ($ref, ref' : \mathbb{P}\,\Sigma$ and $\checkmark \notin \Sigma$)]

$$= \left\{ \begin{array}{l} w, w', Y, Z, s, t \mid (s, Y) \in \{w, w' \mid P \bullet (tr', ref')\} \\ \qquad \wedge (t, Z) \in \{w, w' \mid Q \bullet (tr', ref')\} \\ \qquad \wedge tr' - tr \in (s - tr \parallel_{cs} t - tr) \\ \qquad \wedge s \upharpoonright cs = t \upharpoonright cs \\ \qquad \wedge ref' = Y \cup Z \\ \qquad \wedge Y \setminus \mathsf{cs} = Z \setminus \mathsf{cs} \\ \bullet (tr' - tr, ref') \end{array} \right\}$$

[Lemma J.55]

$$= \left\{ \begin{array}{l} w, w', Y, Z, s, t \mid (s, Y) \in \{w, w' \mid P \bullet (tr', ref')\} \\ \qquad \wedge (t, Z) \in \{w, w' \mid Q \bullet (tr', ref')\} \\ \qquad \wedge tr' - tr \in (s - tr \parallel_{cs} t - tr) \\ \qquad \wedge s \upharpoonright cs = t \upharpoonright cs \\ \qquad \wedge ref' = \left(\begin{array}{l} ((Y \cup Z) \cap cs) \\ \cup((Y \cap Z) \setminus cs) \end{array} \right) \\ \qquad \wedge Y \setminus \mathsf{cs} = Z \setminus \mathsf{cs} \\ \bullet (tr' - tr, ref') \end{array} \right\}$$

Here, we use the fact that ***Circus*** actions are **C2**-healthy, which guarantees that the final sets of refusals ref' are subset closed. This guarantees that the set of values assigned to Y and Z in the production of the set of failures are subset closed. Because of this, we might change the condition in the outermost set comprehension by assuring that ref' is a subset (not necessarily equals) of the previous set expression on Y, Z and cs. Furthermore, we might also drop the condition $Y \setminus \mathsf{cs} = Z \setminus \mathsf{cs}$.

[P and Q are **C2** (ref' is subset closed), SC and ST]

$$= \left\{ \begin{array}{l} w, w', Y, Z, s, t \mid (s, Y) \in \{w, w' \mid P \bullet (tr', ref')\} \\ \qquad \wedge (t, Z) \in \{w, w' \mid Q \bullet (tr', ref')\} \\ \qquad \wedge tr' - tr \in (s - tr \parallel_{cs} t - tr) \\ \qquad \wedge s \upharpoonright cs = t \upharpoonright cs \\ \qquad \wedge ref' \subseteq \left(\begin{array}{l} ((Y \cup Z) \cap cs) \\ \cup((Y \cap Z) \setminus cs) \end{array} \right) \\ \bullet (tr' - tr, ref') \end{array} \right\}$$

[ST, PC and SC]

$$= \left\{ \begin{array}{l} w, w', Y, Z, s, t \mid (s, Y) \in \{w, w' \mid P \bullet (tr', ref')\} \\ \qquad\qquad \wedge\ (t, Z) \in \{w, w' \mid Q \bullet (tr', ref')\} \\ \qquad\qquad \wedge\ tr' - tr \in (s - tr \parallel_{cs} t - tr) \\ \qquad\qquad \wedge\ s \upharpoonright cs = t \upharpoonright cs \\ \qquad\qquad \wedge\ ref' \subseteq \left(\begin{array}{l} ((Y \cup Z) \cap cs) \\ \cup((Y \cap Z) \setminus cs) \end{array} \right) \\ \bullet\ (tr' - tr, ref') \end{array} \right\}$$

[PC and SC]

$$= \left\{ \begin{array}{l} w, w' \mid \exists\, 1.w_0, 2.w_0 \bullet \\ \qquad P[1.w_0/w'] \wedge Q[2.w_0/w'] \\ \qquad \wedge\ tr' - tr \in (1.tr_0 - tr \parallel_{cs} 2.tr_0 - tr) \\ \qquad \wedge\ 1.tr_0 \upharpoonright cs = 2.tr_0 \upharpoonright cs \\ \qquad \wedge\ ref' \subseteq \left(\begin{array}{l} ((1.ref_0 \cup 2.ref_0) \cap cs) \\ \cup((1.ref_0 \cap 2.ref_0) \setminus cs) \end{array} \right) \\ \bullet\ (tr' - tr, ref') \end{array} \right\}$$

[Notation]

$$= \left\{ \begin{array}{l} (tr' - tr, ref') \mid \exists\, 1.w_0, 2.w_0 \bullet \\ \qquad P[1.w_0/w'] \wedge Q[2.w_0/w'] \\ \qquad \wedge\ tr' - tr \in (1.tr_0 - tr \parallel_{cs} 2.tr_0 - tr) \\ \qquad \wedge\ 1.tr_0 \upharpoonright cs = 2.tr_0 \upharpoonright cs \\ \qquad \wedge\ ref' \subseteq \left(\begin{array}{l} ((1.ref_0 \cup 2.ref_0) \cap cs) \\ \cup((1.ref_0 \cap 2.ref_0) \setminus cs) \end{array} \right) \end{array} \right\}$$

Lemma J.37

$$\{(tr' - tr, ref') \mid okay \wedge wait' \wedge (PQ;\ M_{P_t Q_t})\}$$
$$=$$
$$\left\{ \begin{array}{l} (u, Y \cup Z) \\ \mid\ Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \ \wedge\ \exists\, s, t \bullet (s, Y) \in \{(tr' - tr, ref') \mid okay \wedge wait' \wedge (P)^t_f\} \\ \qquad\qquad \wedge\ (t, Z) \in \{(tr' - tr, ref') \mid okay \wedge wait' \wedge (Q)^t_f\} \\ \qquad\qquad \wedge\ u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array} \right\}$$

Proof.

$$\{(tr' - tr, ref') \mid okay \wedge wait' \wedge (PQ;\ M_{P_t Q_t})\}$$

[PQ and $M_{P_t Q_t}$]

$$=$$

$$\left\{\begin{array}{l} (tr' - tr, ref') \mid \\ \qquad okay \wedge wait' \\ \qquad \wedge \left(\begin{array}{l} \left(\begin{array}{l} (P^t_f;\ U1(out\alpha\, P)) \\ \wedge\, (Q^t_f;\ U2(out\alpha\, Q)) \end{array}\right)_{+\{v,tr\}} ; \\ \left(\begin{array}{l} tr' - tr \in (1.tr - tr \parallel_{cs} 2.tr - tr) \\ \wedge\, 1.tr \upharpoonright cs = 2.tr \upharpoonright cs \\ \wedge\, (1.wait \wedge 2.wait) \\ \wedge\, ref' \subseteq \left(\begin{array}{l} ((1.ref \cup 2.ref) \cap cs) \\ \cup((1.ref \cap 2.ref) \setminus cs) \end{array}\right) \end{array}\right) \end{array}\right) \end{array}\right\}$$

[Case Analysis on $wait'$ and PC]

$$= \left\{\begin{array}{l} (tr' - tr, ref') \mid \\ \qquad okay \\ \qquad \wedge \left(\begin{array}{l} \left(\begin{array}{l} (P^t_f;\ U1(out\alpha\, P)) \\ \wedge\, (Q^t_f;\ U2(out\alpha\, Q)) \end{array}\right)_{+\{v,tr\}} ; \\ \left(\begin{array}{l} tr' - tr \in (1.tr - tr \parallel_{cs} 2.tr - tr) \\ \wedge\, 1.tr \upharpoonright cs = 2.tr \upharpoonright cs \\ \wedge\, (1.wait \wedge 2.wait) \\ \wedge\, ref' \subseteq \left(\begin{array}{l} ((1.ref \cup 2.ref) \cap cs) \\ \cup((1.ref \cap 2.ref) \setminus cs) \end{array}\right) \end{array}\right) \end{array}\right) \end{array}\right\}$$

[U_n, $A_{+\{v,tr\}}$ and Sequence Composition]

$$= \left\{\begin{array}{l} (tr' - tr, ref') \mid \\ \qquad okay \\ \qquad \wedge \left(\begin{array}{l} \left(\begin{array}{l} \left(P^t_f \left[\begin{array}{l} 1.okay', 1.wait', 1.tr', 1.ref', 1.v'/ \\ okay', wait', tr', ref', v' \end{array}\right]\right) \\ \wedge \left(Q^t_f \left[\begin{array}{l} 2.okay', 2.wait', 2.tr', 2.ref', 2.v'/ \\ okay', wait', tr', ref', v' \end{array}\right]\right) \\ v' = v \wedge tr' = tr \end{array}\right) ; \\ \left(\begin{array}{l} tr' - tr \in (1.tr - tr \parallel_{cs} 2.tr - tr) \\ \wedge\, 1.tr \upharpoonright cs = 2.tr \upharpoonright cs \\ \wedge\, (1.wait \wedge 2.wait) \\ \wedge\, ref' \subseteq \left(\begin{array}{l} ((1.ref \cup 2.ref) \cap cs) \\ \cup((1.ref \cap 2.ref) \setminus cs) \end{array}\right) \end{array}\right) \end{array}\right) \end{array}\right\}$$

[Sequence Composition]

$$= \left\{ \begin{array}{l} (tr' - tr, ref') \mid \\ \qquad okay \\ \qquad \wedge \left(\begin{array}{l} \exists\, okay_0, wait_0, tr_0, ref_0, v_0, \\ \quad 1.okay_0, 1.wait_0, 1.tr_0, 1.ref_0, 1.v_0, \\ \quad 2.okay_0, 2.wait_0, 2.tr_0, 2.ref_0, 2.v_0 \bullet \\ \qquad \left(P^t_f \left[\begin{array}{l} 1.okay_0, 1.wait_0, 1.tr_0, 1.ref_0, 1.v_0 / \\ okay', wait', tr', ref', v' \end{array} \right] \right) \\ \qquad \wedge \left(Q^t_f \left[\begin{array}{l} 2.okay_0, 2.wait_0, 2.tr_0, 2.ref_0, 2.v_0 / \\ okay', wait', tr', ref', v' \end{array} \right] \right) \\ \qquad \wedge\, v_0 = v \wedge tr_0 = tr \\ \qquad \wedge\, tr' - tr_0 \in (1.tr_0 - tr_0 \parallel_{cs} 2.tr_0 - tr_0) \\ \qquad \wedge\, 1.tr_0 \upharpoonright cs = 2.tr_0 \upharpoonright cs \\ \qquad \wedge\, (1.wait_0 \wedge 2.wait_0) \\ \qquad \wedge\, ref' \subseteq \left(\begin{array}{l} ((1.ref_0 \cup 2.ref_0) \cap cs) \\ \cup((1.ref_0 \cap 2.ref_0) \setminus cs) \end{array} \right) \end{array} \right) \end{array} \right\}$$

[PC]

$$= \left\{ \begin{array}{l} (tr' - tr, ref') \mid \\ \qquad okay \\ \qquad \wedge \left(\begin{array}{l} \exists\, 1.okay_0, 1.wait_0, 1.tr_0, 1.ref_0, 1.v_0, \\ \quad 2.okay_0, 2.wait_0, 2.tr_0, 2.ref_0, 2.v_0 \bullet \\ \qquad \left(P^t_f \left[\begin{array}{l} 1.okay_0, 1.wait_0, 1.tr_0, 1.ref_0, 1.v_0 / \\ okay', wait', tr', ref', v' \end{array} \right] \right) \\ \qquad \wedge \left(Q^t_f \left[\begin{array}{l} 2.okay_0, 2.wait_0, 2.tr_0, 2.ref_0, 2.v_0 / \\ okay', wait', tr', ref', v' \end{array} \right] \right) \\ \qquad \wedge\, tr' - tr \in (1.tr_0 - tr \parallel_{cs} 2.tr_0 - tr) \\ \qquad \wedge\, 1.tr_0 \upharpoonright cs = 2.tr_0 \upharpoonright cs \\ \qquad \wedge\, (1.wait_0 \wedge 2.wait_0) \\ \qquad \wedge\, ref' \subseteq \left(\begin{array}{l} ((1.ref_0 \cup 2.ref_0) \cap cs) \\ \cup((1.ref_0 \cap 2.ref_0) \setminus cs) \end{array} \right) \end{array} \right) \end{array} \right\}$$

[PC]

$$= \left\{ \begin{array}{l} (tr' - tr, ref') \mid \\ \quad \exists\, 1.okay_0, 1.wait_0, 1.tr_0, 1.ref_0, 1.v_0, \\ \quad\; 2.okay_0, 2.wait_0, 2.tr_0, 2.ref_0, 2.v_0 \bullet \\ \qquad \left((okay \land wait' \land P^t_f) \left[\begin{array}{l} 1.okay_0, 1.wait_0, 1.tr_0, 1.ref_0, 1.v_0 / \\ okay', wait', tr', ref', v' \end{array} \right] \right) \\ \qquad \land \left((okay \land wait' \land Q^t_f) \left[\begin{array}{l} 2.okay_0, 2.wait_0, 2.tr_0, 2.ref_0, 2.v_0 / \\ okay', wait', tr', ref', v' \end{array} \right] \right) \\ \qquad \land\, tr' - tr \in (1.tr_0 - tr \parallel_{cs} 2.tr_0 - tr) \\ \qquad \land\, 1.tr_0 \upharpoonright cs = 2.tr_0 \upharpoonright cs \\ \qquad \land\, ref' \subseteq \left(\begin{array}{l} ((1.ref_0 \cup 2.ref_0) \cap cs) \\ \cup((1.ref_0 \cap 2.ref_0) \setminus cs) \end{array} \right) \end{array} \right\}$$

[Lemma J.36]

$$= \left\{ \begin{array}{l} (u, Y \cup Z) \\ \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \;\land \exists\, s, t \bullet (s, Y) \in \{(tr' - tr, ref') \mid okay \land wait' \land (P)^t_f\} \\ \qquad\qquad \land (t, Z) \in \{(tr' - tr, ref') \mid okay \land wait' \land (Q)^t_f\} \\ \qquad\qquad \land u \in s \parallel_{cs} t \\ \qquad\qquad \land s \upharpoonright cs = t \upharpoonright cs \end{array} \right\}$$

[Lemma J.50]

$$= \left\{ \begin{array}{l} (u, Y \cup Z) \\ \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \;\land \exists\, s, t \bullet (s, Y) \in \{(tr' - tr, ref') \mid okay \land wait' \land (P)^t_f\} \\ \qquad\qquad \land (t, Z) \in \{(tr' - tr, ref') \mid okay \land wait' \land (Q)^t_f\} \\ \qquad\qquad \land u \in s \underset{\mathsf{cs}}{\parallel} t \\ \qquad\qquad \land s \upharpoonright cs = t \upharpoonright cs \end{array} \right\}$$

[ST, Lemma J.52 and PC]

$$= \left\{ \begin{array}{l} (u, Y \cup Z) \\ \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \;\land \exists\, s, t \bullet (s, Y) \in \{(tr' - tr, ref') \mid okay \land wait' \land (P)^t_f\} \\ \qquad\qquad \land (t, Z) \in \{(tr' - tr, ref') \mid okay \land wait' \land (Q)^t_f\} \\ \qquad\qquad \land u \in s \underset{\mathsf{cs}}{\parallel} t \end{array} \right\}$$

[Lemma J.51 ($tr, tr' : \mathrm{seq}\,\Sigma$ and $\checkmark \notin \Sigma$)]

$$= \left\{ \begin{array}{l} (u, Y \cup Z) \\ \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \quad \wedge \exists\, s, t \bullet (s, Y) \in \{(tr' - tr, ref') \mid okay \wedge wait' \wedge (P)^t_f\} \\ \qquad\qquad \wedge (t, Z) \in \{(tr' - tr, ref') \mid okay \wedge wait' \wedge (Q)^t_f\} \\ \qquad\qquad \wedge u \in s \underset{\mathsf{cs}^\checkmark}{\parallel} t \end{array} \right\}$$

Lemma J.38

$$\begin{array}{l} \{(tr' - tr, ref') \mid okay \wedge wait' \wedge (PQ;\, M_{P_t Q_f})\} \\ = \\ \left\{ \begin{array}{l} (u, Y \cup Z) \\ \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \quad \wedge \exists\, s, t \bullet (s, Y) \in \{(tr' - tr, ref') \mid okay \wedge wait' \wedge (P)^t_f\} \\ \qquad\qquad \wedge (t, Z) \in \{(tr' - tr, ref') \mid okay \wedge \neg\, wait' \wedge (Q)^t_f\} \\ \qquad\qquad \wedge u \in s \underset{\mathsf{cs}^\checkmark}{\parallel} t \end{array} \right\} \end{array}$$

Proof. Very similar to that of Lemma J.37.

Lemma J.39

$$\begin{array}{l} \{(tr' - tr, ref') \mid okay \wedge wait' \wedge (PQ;\, M_{P_f Q_t})\} \\ = \\ \left\{ \begin{array}{l} (u, Y \cup Z) \\ \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \quad \wedge \exists\, s, t \bullet (s, Y) \in \{(tr' - tr, ref') \mid okay \wedge \neg\, wait' \wedge (P)^t_f\} \\ \qquad\qquad \wedge (t, Z) \in \{(tr' - tr, ref') \mid okay \wedge wait' \wedge (Q)^t_f\} \\ \qquad\qquad \wedge u \in s \underset{\mathsf{cs}^\checkmark}{\parallel} t \end{array} \right\} \end{array}$$

Proof. Very similar to that of Lemma J.37.

Lemma J.40

$$\begin{array}{l} \{(tr' - tr, ref') \mid okay \wedge \neg\, wait' \wedge (PQ;\, M_{P_f Q_f})\} \\ = \\ \left\{ \begin{array}{l} (u, Y \cup Z) \\ \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \quad \wedge \exists\, s, t \bullet (s, Y) \in \{(tr' - tr, ref') \mid okay \wedge \neg\, wait' \wedge (P)^t_f\} \\ \qquad\qquad \wedge (t, Z) \in \{(tr' - tr, ref') \mid okay \wedge \neg\, wait' \wedge (Q)^t_f\} \\ \qquad\qquad \wedge u \in s \underset{\mathsf{cs}^\checkmark}{\parallel} t \end{array} \right\} \end{array}$$

Proof. Very similar to that of Lemma J.37.

Lemma J.41

$$\{(tr' - tr, ref' \cup \{\checkmark\}) \mid okay \wedge wait' \wedge (PQ;\ M_{P_t Q_t})\}$$
$$=$$
$$\left\{\begin{array}{l} (u, Y \cup Z \cup \{\checkmark\}) \\ \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \quad \wedge \exists\, s, t \bullet (s, Y) \in \{(tr' - tr, ref') \mid okay \wedge wait' \wedge (P)^t_f\} \\ \qquad\qquad \wedge (t, Z) \in \{(tr' - tr, ref') \mid okay \wedge wait' \wedge (Q)^t_f\} \\ \qquad\qquad \wedge u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array}\right\}$$

Proof.

$$\{(tr' - tr, ref' \cup \{\checkmark\}) \mid okay \wedge wait' \wedge (PQ;\ M_{P_t Q_t})\} \qquad [\text{SC}]$$
$$=$$
$$\{(t, r \cup \{\checkmark\}) \mid (t, r) \in \{(tr' - tr, ref') \mid okay \wedge wait' \wedge (PQ;\ M_{P_t Q_t})\}\} \qquad [\text{Lemma J.37}]$$
$$=$$
$$\left\{\begin{array}{l} (t, r \cup \{\checkmark\}) \\ \mid (t, r) \in \left\{\begin{array}{l} (u, Y \cup Z) \\ \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \quad \wedge \exists\, s, t \bullet (s, Y) \in \{(tr' - tr, ref') \mid okay \wedge wait' \wedge (P)^t_f\} \\ \qquad\qquad \wedge (t, Z) \in \{(tr' - tr, ref') \mid okay \wedge wait' \wedge (Q)^t_f\} \\ \qquad\qquad \wedge u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array}\right\} \end{array}\right\} \qquad [\text{SC}]$$
$$= \left\{\begin{array}{l} (u, Y \cup Z \cup \{\checkmark\}) \\ \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \quad \wedge \exists\, s, t \bullet (s, Y) \in \{(tr' - tr, ref') \mid okay \wedge wait' \wedge (P)^t_f\} \\ \qquad\qquad \wedge (t, Z) \in \{(tr' - tr, ref') \mid okay \wedge wait' \wedge (Q)^t_f\} \\ \qquad\qquad \wedge u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array}\right\}$$

Lemma J.42

$$\{(tr' - tr, ref' \cup \{\checkmark\}) \mid okay \land wait' \land (PQ;\ M_{P_t Q_f})\}$$
$$=$$
$$\left\{\begin{array}{l} (u, Y \cup Z \cup \{\checkmark\}) \\ \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \quad \land \exists\, s, t \bullet (s, Y) \in \{(tr' - tr, ref') \mid okay \land wait' \land (P)^t_f\} \\ \qquad\qquad \land (t, Z) \in \{(tr' - tr, ref') \mid okay \land \neg\, wait' \land (Q)^t_f\} \\ \qquad\qquad \land u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array}\right\}$$

Proof.

$$\{(tr' - tr, ref' \cup \{\checkmark\}) \mid okay \land wait' \land (PQ;\ M_{P_t Q_f})\} \qquad \text{[SC]}$$
$$=$$
$$\{(t, r \cup \{\checkmark\}) \mid (t, r) \in \{(tr' - tr, ref') \mid okay \land wait' \land (PQ;\ M_{P_t Q_f})\}\}$$
[Lemma J.38]
$$=$$
$$\left\{\begin{array}{l} (t, r \cup \{\checkmark\}) \\ \mid (t, r) \in \left\{\begin{array}{l} (u, Y \cup Z) \\ \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \quad \land \exists\, s, t \bullet (s, Y) \in \{(tr' - tr, ref') \mid okay \land wait' \land (P)^t_f\} \\ \qquad\qquad \land (t, Z) \in \{(tr' - tr, ref') \mid okay \land \neg\, wait' \land (Q)^t_f\} \\ \qquad\qquad \land u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array}\right\} \end{array}\right\}$$
[SC]
$$= \left\{\begin{array}{l} (u, Y \cup Z \cup \{\checkmark\}) \\ \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \quad \land \exists\, s, t \bullet (s, Y) \in \{(tr' - tr, ref') \mid okay \land wait' \land (P)^t_f\} \\ \qquad\qquad \land (t, Z) \in \{(tr' - tr, ref') \mid okay \land \neg\, wait' \land (Q)^t_f\} \\ \qquad\qquad \land u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array}\right\}$$

Lemma J.43

$$
\begin{array}{l}
\{(tr' - tr, ref' \cup \{\checkmark\}) \mid okay \land wait' \land (PQ;\ M_{P_f\,Q_t})\} \\
= \\
\left\{
\begin{array}{l}
(u, Y \cup Z \cup \{\checkmark\}) \\
\mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\
\quad \land \exists\, s, t \bullet (s, Y) \in \{(tr' - tr, ref') \mid okay \land \neg\, wait' \land (P)^t_f\} \\
\qquad\qquad \land (t, Z) \in \{(tr' - tr, ref') \mid okay \land wait' \land (Q)^t_f\} \\
\qquad\qquad \land u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t
\end{array}
\right\}
\end{array}
$$

Proof.

$$
\begin{array}{lr}
\{(tr' - tr, ref' \cup \{\checkmark\}) \mid okay \land wait' \land (PQ;\ M_{P_f\,Q_t})\} & [\text{SC}] \\
= & \\
\{(t, r \cup \{\checkmark\}) \mid (t, r) \in \{(tr' - tr, ref') \mid okay \land wait' \land (PQ;\ M_{P_f\,Q_t})\}\} & \\
& [\text{Lemma J.39}] \\
= & \\
\left\{
\begin{array}{l}
(t, r \cup \{\checkmark\}) \\
\mid (t, r) \in \left\{
\begin{array}{l}
(u, Y \cup Z) \\
\mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\
\quad \land \exists\, s, t \bullet (s, Y) \in \{(tr' - tr, ref') \mid okay \land \neg\, wait' \land (P)^t_f\} \\
\qquad\qquad \land (t, Z) \in \{(tr' - tr, ref') \mid okay \land wait' \land (Q)^t_f\} \\
\qquad\qquad \land u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t
\end{array}
\right\}
\end{array}
\right\} & \\
& [\text{SC}] \\
= \left\{
\begin{array}{l}
(u, Y \cup Z \cup \{\checkmark\}) \\
\mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\
\quad \land \exists\, s, t \bullet (s, Y) \in \{(tr' - tr, ref') \mid okay \land \neg\, wait' \land (P)^t_f\} \\
\qquad\qquad \land (t, Z) \in \{(tr' - tr, ref') \mid okay \land wait' \land (Q)^t_f\} \\
\qquad\qquad \land u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t
\end{array}
\right\} &
\end{array}
$$

Lemma J.44

$$\{((tr' - tr) \frown \langle \checkmark \rangle, ref') \mid okay \land wait' \land (PQ;\, M_{P_f Q_f})\}$$
$$=$$
$$\left\{\begin{array}{l}(u \frown \langle \checkmark \rangle, Y \cup Z)\\ \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\})\\ \quad \land \exists\, s, t \bullet (s, Y) \in \{(tr' - tr, ref') \mid okay \land \neg\, wait' \land (P)^t_f\}\\ \qquad\qquad \land (t, Z) \in \{(tr' - tr, ref') \mid okay \land wait' \land (Q)^t_f\}\\ \qquad\qquad \land u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t\end{array}\right\}$$

Proof.

$$\{((tr' - tr) \frown \langle \checkmark \rangle, ref') \mid okay \land wait' \land (PQ;\, M_{P_f Q_f})\} \qquad \text{[SC]}$$
$$=$$
$$\{(t \frown \langle \checkmark \rangle, r) \mid (t, r) \in \{(tr' - tr, ref') \mid okay \land wait' \land (PQ;\, M_{P_f Q_f})\}\}$$
$$\text{[Lemma J.40]}$$
$$=$$
$$\left\{\begin{array}{l}(t \frown \langle \checkmark \rangle, r)\\ \mid (t, r) \in \left\{\begin{array}{l}(u, Y \cup Z)\\ \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\})\\ \quad \land \exists\, s, t \bullet (s, Y) \in \{(tr' - tr, ref') \mid okay \land \neg\, wait' \land (P)^t_f\}\\ \qquad\qquad \land (t, Z) \in \{(tr' - tr, ref') \mid okay \land \neg\, wait' \land (Q)^t_f\}\\ \qquad\qquad \land u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t\end{array}\right\}\end{array}\right\}$$
$$\text{[SC]}$$
$$= \left\{\begin{array}{l}(u \frown \langle \checkmark \rangle, Y \cup Z)\\ \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\})\\ \quad \land \exists\, s, t \bullet (s, Y) \in \{(tr' - tr, ref') \mid okay \land \neg\, wait' \land (P)^t_f\}\\ \qquad\qquad \land (t, Z) \in \{(tr' - tr, ref') \mid okay \land \neg\, wait' \land (Q)^t_f\}\\ \qquad\qquad \land u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t\end{array}\right\}$$

Lemma J.45

$$
\begin{array}{l}
\{((tr' - tr) \frown \langle \checkmark \rangle, ref' \cup \{\checkmark\}) \mid okay \land wait' \land (PQ;\ M_{P_f Q_f})\} \\
= \\
\left\{
\begin{array}{l}
(u \frown \langle \checkmark \rangle, Y \cup Z \cup \{\checkmark\}) \\
\mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\
\quad \land \exists\, s, t \bullet (s, Y) \in \{(tr' - tr, ref') \mid okay \land \neg\ wait' \land (P)^t_f\} \\
\qquad\qquad \land (t, Z) \in \{(tr' - tr, ref') \mid okay \land wait' \land (Q)^t_f\} \\
\qquad\qquad \land u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t
\end{array}
\right\}
\end{array}
$$

Proof.

$$
\begin{array}{ll}
\{((tr' - tr) \frown \langle \checkmark \rangle, ref' \cup \{\checkmark\}) \mid okay \land wait' \land (PQ;\ M_{P_f Q_f})\} & [\text{SC}] \\
= & \\
\{(t \frown \langle \checkmark \rangle, r \cup \{\checkmark\}) \mid (t, r) \in \{(tr' - tr, ref') \mid okay \land wait' \land (PQ;\ M_{P_f Q_f})\}\} & \\
& [\text{Lemma J.40}] \\
= &
\end{array}
$$

$$
\left\{
\begin{array}{l}
(t \frown \langle \checkmark \rangle, r \cup \{\checkmark\}) \\
\mid (t, r) \in \left\{
\begin{array}{l}
(u, Y \cup Z) \\
\mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\
\quad \land \exists\, s, t \bullet (s, Y) \in \{(tr' - tr, ref') \mid okay \land \neg\ wait' \land (P)^t_f\} \\
\qquad\qquad \land (t, Z) \in \{(tr' - tr, ref') \mid okay \land \neg\ wait' \land (Q)^t_f\} \\
\qquad\qquad \land u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t
\end{array}
\right\}
\end{array}
\right\}
$$

[SC]

$$
= \left\{
\begin{array}{l}
(u \frown \langle \checkmark \rangle, Y \cup Z \cup \{\checkmark\}) \\
\mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\
\quad \land \exists\, s, t \bullet (s, Y) \in \{(tr' - tr, ref') \mid okay \land \neg\ wait' \land (P)^t_f\} \\
\qquad\qquad \land (t, Z) \in \{(tr' - tr, ref') \mid okay \land \neg\ wait' \land (Q)^t_f\} \\
\qquad\qquad \land u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t
\end{array}
\right\}
$$

J.4 Theorems

All theorems use the operation $tr' - tr$. This is only well defined for $tr\ prefix\ tr'$. Hence, all theorems below implicitly require the actions involved to be **R1**.

Theorem J.2 $traces^{\mathcal{UTP}}(Skip) = traces(\Upsilon(Skip))$

Proof.

$$\begin{array}{ll}
\mathit{traces}^{\mathcal{UTP}}(\mathit{Skip}) & [\mathit{traces}^{\mathcal{UTP}}] \\
= \{tr' - tr \mid (\mathit{Skip})^n\} & [A^t] \\
\quad \cup \{(tr' - tr) \frown \langle \checkmark \rangle \mid (\mathit{Skip})^t\} & \\
= \{tr' - tr \mid (\mathit{Skip})^n\} & [A^n] \\
\quad \cup \{(tr' - tr) \frown \langle \checkmark \rangle \mid \neg\, wait' \land (\mathit{Skip})^n\} & \\
= \{tr' - tr \mid okay \land \neg\, wait \land okay' \land \mathit{Skip}\} & [\text{PC}] \\
\quad \cup \{(tr' - tr) \frown \langle \checkmark \rangle \mid okay \land \neg\, wait \land okay' \land \neg\, wait' \land \mathit{Skip}\} & \\
= \{tr' - tr \mid okay \land (\mathit{Skip})^t_f\} & [\text{Lemma J.12}] \\
\quad \cup \{(tr' - tr) \frown \langle \checkmark \rangle \mid okay \land \neg\, wait' \land (\mathit{Skip})^t_f\} & \\
= \{tr' - tr \mid okay \land \mathbf{CSP1}(tr' = tr \land \neg\, wait' \land v' = v)\} & [\text{Lemma J.4}] \\
\quad \cup \{(tr' - tr) \frown \langle \checkmark \rangle \mid okay \land \neg\, wait' \land \mathbf{CSP1}(tr' = tr \land \neg\, wait' \land v' = v)\} & \\
= \{tr' - tr \mid okay \land tr' = tr \land \neg\, wait' \land v' = v\} & [\text{PC}] \\
\quad \cup \{(tr' - tr) \frown \langle \checkmark \rangle \mid okay \land \neg\, wait' \land tr' = tr \land \neg\, wait' \land v' = v\} & \\
= \{tr' - tr \mid okay \land tr' = tr \land \neg\, wait' \land v' = v\} & [\text{SS. and } -] \\
\quad \cup \{(tr' - tr) \frown \langle \checkmark \rangle \mid okay \land tr' = tr \land \neg\, wait' \land v' = v\} & \\
= \{\langle\rangle \mid okay \land tr' = tr \land \neg\, wait' \land v' = v\} & [\text{Cases and SC}] \\
\quad \cup \{\langle \checkmark \rangle \mid okay \land tr' = tr \land \neg\, wait' \land v' = v\} & \\
= \{\langle\rangle\} \cup \{\} \cup \{\langle \checkmark \rangle\} \cup \{\} & [\text{ST}] \\
= \{\langle\rangle, \langle \checkmark \rangle\} & [\mathit{traces}] \\
= \mathit{traces}(\texttt{SKIP}) & [\Upsilon] \\
= \mathit{traces}(\Upsilon(\mathit{Skip})) &
\end{array}$$

Theorem J.3 $\mathit{traces}^{\mathcal{UTP}}(\mathit{Stop}) = \mathit{traces}(\Upsilon(\mathit{Stop}))$

Proof.

$$\begin{array}{ll}
\mathit{traces}^{\mathcal{UTP}}(\mathit{Stop}) & [\mathit{traces}^{\mathcal{UTP}}] \\
= \{tr' - tr \mid (\mathit{Stop})^n\} & [A^t] \\
\quad \cup \{(tr' - tr) \frown \langle \checkmark \rangle \mid (\mathit{Stop})^t\} & \\
= \{tr' - tr \mid (\mathit{Stop})^n\} & [A^n] \\
\quad \cup \{(tr' - tr) \frown \langle \checkmark \rangle \mid \neg\, wait' \land (\mathit{Stop})^n\} & \\
= \{tr' - tr \mid ok \land \neg\, wait \land ok' \land \mathit{Stop}\} & [\text{PC}] \\
\quad \cup \{(tr' - tr) \frown \langle \checkmark \rangle \mid ok \land \neg\, wait \land ok' \land \neg\, wait' \land \mathit{Stop}\} &
\end{array}$$

$$
\begin{aligned}
&= \{tr' - tr \mid ok \land (Stop)_f^t\} && \text{[Lemma J.13]}\\
&\quad \cup \{(tr' - tr) ^\frown \langle \checkmark \rangle \mid ok \land \neg\, wait' \land (Stop)_f^t\}\\
&= \{tr' - tr \mid ok \land \mathbf{CSP1}(tr' = tr \land wait')\} && \text{[Lemma J.4]}\\
&\quad \cup \{(tr' - tr) ^\frown \langle \checkmark \rangle \mid ok \land \neg\, wait' \land \mathbf{CSP1}(tr' = tr \land wait')\}\\
&= \{tr' - tr \mid ok \land tr' = tr \land wait'\} && \text{[PC]}\\
&\quad \cup \{(tr' - tr) ^\frown \langle \checkmark \rangle \mid ok \land \neg\, wait' \land tr' = tr \land wait'\}\\
&= \{tr' - tr \mid ok \land tr' = tr \land wait'\} && \text{[SS. and } -]\\
&\quad \cup \{(tr' - tr) ^\frown \langle \checkmark \rangle \mid false\}\\
&= \{\langle\rangle \mid ok \land tr' = tr \land \neg\, wait' \land v' = v\}\\
&\quad \cup \{\langle \checkmark \rangle \mid false\}\\
& && \text{[Cases and SC]}\\
&= \{\langle\rangle\} \cup \{\} \cup \{\} && \text{[ST]}\\
&= \{\langle\rangle\} && [traces]\\
&= traces(\texttt{STOP}) && [\Upsilon]\\
&= traces(\Upsilon(Stop))
\end{aligned}
$$

Theorem J.4 $traces^{\mathcal{UTP}}(c \to Skip) = traces(\Upsilon(c \to Skip))$

Proof.

$$
\begin{aligned}
&traces^{\mathcal{UTP}}(c \to Skip) && [traces^{\mathcal{UTP}}]\\
&= \{tr' - tr \mid (c \to Skip)^n\} && [A^t]\\
&\quad \cup \{(tr' - tr) ^\frown \langle \checkmark \rangle \mid (c \to Skip)^t\}\\
&= \{tr' - tr \mid (c \to Skip)^n\} && [A^n]\\
&\quad \cup \{(tr' - tr) ^\frown \langle \checkmark \rangle \mid \neg\, wait' \land (c \to Skip)^n\}\\
&= \{tr' - tr \mid okay \land \neg\, wait \land okay' \land c \to Skip\} && \text{[PC]}\\
&\quad \cup \{(tr' - tr) ^\frown \langle \checkmark \rangle \mid okay \land \neg\, wait \land okay' \land \neg\, wait' \land c \to Skip\}\\
&= \{tr' - tr \mid okay \land (c \to Skip)_f^t\} && \text{[PC]}\\
&\quad \cup \{(tr' - tr) ^\frown \langle \checkmark \rangle \mid okay \land \neg\, wait' \land (c \to Skip)_f^t\}\\
&= \{tr' - tr \mid okay \land ((c \to Skip)_f)^t\} && \text{[Lemma J.3]}\\
&\quad \cup \{(tr' - tr) ^\frown \langle \checkmark \rangle \mid okay \land \neg\, wait' \land ((c \to Skip)_f)^t\}\\
&= \{tr' - tr \mid okay \land (\mathbf{CSP1}(okay' \land do_{\mathcal{C}}(c, Sync) \land v' = v))^t\} && \text{[PC]}\\
&\quad \cup \{(tr' - tr) ^\frown \langle \checkmark \rangle \mid okay \land \neg\, wait' \land (\mathbf{CSP1}(okay' \land do_{\mathcal{C}}(c, Sync) \land v' = v))^t\}\\
&= \{tr' - tr \mid (okay \land \mathbf{CSP1}(okay' \land do_{\mathcal{C}}(c, Sync) \land v' = v))^t\} && \text{[Lemma J.4]}\\
&\quad \cup \{(tr' - tr) ^\frown \langle \checkmark \rangle \mid (okay \land \neg\, wait' \land \mathbf{CSP1}(okay' \land do_{\mathcal{C}}(c, Sync) \land v' = v))^t\}
\end{aligned}
$$

$$
\begin{array}{ll}
= \{tr' - tr \mid (okay \land okay' \land do_{\mathcal{C}}(c, Sync) \land v' = v)^t\} & [\text{PC}] \\
\quad \cup \{(tr' - tr) ^\frown \langle \checkmark \rangle \mid (okay \land \neg\, wait' \land okay' \land do_{\mathcal{C}}(c, Sync) \land v' = v)^t\} & \\
= \{tr' - tr \mid okay \land do_{\mathcal{C}}(c, Sync) \land v' = v\} & [\text{PC and } do_{\mathcal{C}}] \\
\quad \cup \{(tr' - tr) ^\frown \langle \checkmark \rangle \mid okay \land \neg\, wait' \land do_{\mathcal{C}}(c, Sync) \land v' = v\} & \\
= \left\{ \begin{array}{l} tr' - tr \mid okay \land v' = v \land \\ \qquad (tr' = tr \land (c, Sync) \notin ref' \lhd wait' \rhd tr' = tr ^\frown \langle (c, Sync) \rangle) \end{array} \right\} & \\
\quad \cup \left\{ \begin{array}{l} (tr' - tr) ^\frown \langle \checkmark \rangle \mid okay \land \neg\, wait' \land v' = v \land \\ \qquad (tr' = tr \land (c, Sync) \notin ref' \lhd wait' \rhd tr' = tr ^\frown \langle (c, Sync) \rangle) \end{array} \right\} & \\
 & [\text{PC and } do_{\mathcal{C}}] \\
= \{tr' - tr \mid okay \land v' = v \land wait' \land tr' = tr \land (c, Sync) \notin ref'\} & [\text{SS. and } -] \\
\quad \cup \{tr' - tr \mid okay \land v' = v \land \neg\, wait' \land tr' = tr ^\frown \langle (c, Sync) \rangle\} & \\
\quad \cup \{(tr' - tr) ^\frown \langle \checkmark \rangle \mid okay \land \neg\, wait' \land v' = v \land tr' = tr ^\frown \langle (c, Sync) \rangle\} & \\
= \{\langle\rangle \mid okay \land v' = v \land wait' \land tr' = tr \land (c, Sync) \notin ref'\} & \\
\quad \cup \{\langle (c, Sync) \rangle \mid okay \land v' = v \land \neg\, wait' \land tr' = tr ^\frown \langle (c, Sync) \rangle\} & \\
\quad \cup \{\langle (c, Sync), \checkmark \rangle \mid okay \land \neg\, wait' \land v' = v \land tr' = tr ^\frown \langle (c, Sync) \rangle\} & \\
 & [\textit{Circus} \text{ Events } (c \equiv (c, Sync))] \\
= \{\langle\rangle \mid okay \land v' = v \land wait' \land tr' = tr \land c \notin ref'\} & [\text{Cases and SC}] \\
\quad \cup \{\langle c \rangle \mid okay \land v' = v \land \neg\, wait' \land tr' = tr ^\frown \langle c \rangle\} & \\
\quad \cup \{\langle c, \checkmark \rangle \mid okay \land \neg\, wait' \land v' = v \land tr' = tr ^\frown \langle c \rangle\} & \\
= \{\langle\rangle\} \cup \{\} & [\text{ST}] \\
\quad \cup \{\langle c \rangle\} \cup \{\} & \\
\quad \cup \{\langle c, \checkmark \rangle\} \cup \{\} & \\
= \{\langle\rangle, \langle c \rangle, \langle c, \checkmark \rangle\} & [\text{ST}] \\
= \{\langle\rangle\} \cup \{\langle c \rangle, \langle c, \checkmark \rangle\} & [\text{SC and } ^\frown] \\
= \{\langle\rangle\} \cup \{\langle c \rangle ^\frown s \mid s \in \{\langle\rangle, \langle \checkmark \rangle\}\} & [traces] \\
= \{\langle\rangle\} \cup \{\langle c \rangle ^\frown s \mid s \in traces(\mathtt{SKIP})\} & [traces] \\
= traces(\mathtt{c} \to \mathtt{SKIP}) & [\Upsilon] \\
= traces(\Upsilon(c \to Skip)) &
\end{array}
$$

Theorem J.5 $traces^{\mathcal{UTP}}(c \to A) = traces(\Upsilon(c \to A))$
provided $c \to A$ *is* $\boldsymbol{R}$

Inductive Hypothesis:

$$traces^{\mathcal{UTP}}(A) = traces(\Upsilon(A))$$

Proof.

$$
\begin{array}{ll}
traces^{\mathcal{UTP}}(c \to A) & [traces^{\mathcal{UTP}}] \\
= \{tr' - tr \mid (c \to A)^n\} & [A^t] \\
\quad \cup \{(tr' - tr) \frown \langle \checkmark \rangle \mid (c \to A)^t\} & \\
= \{tr' - tr \mid (c \to A)^n\} & [A^n] \\
\quad \cup \{(tr' - tr) \frown \langle \checkmark \rangle \mid \neg\, wait' \wedge (c \to A)^n\} & \\
= \{tr' - tr \mid okay \wedge \neg\, wait \wedge okay' \wedge c \to A\} & [\text{PC}] \\
\quad \cup \{(tr' - tr) \frown \langle \checkmark \rangle \mid okay \wedge \neg\, wait \wedge okay' \wedge \neg\, wait' \wedge c \to A\} & \\
= \{tr' - tr \mid okay \wedge (c \to A)^t_f\} & [\text{Lemma J.6}] \\
\quad \cup \{(tr' - tr) \frown \langle \checkmark \rangle \mid okay \wedge \neg\, wait' \wedge (c \to A)^t_f\} & \\
= \{tr' - tr \mid okay \wedge (\mathbf{CSP1}(okay' \wedge do_{\mathcal{C}}(c, Sync) \wedge v' = v);\ (A)^t)\} & \\
\quad \cup \{(tr' - tr) \frown \langle \checkmark \rangle \mid & \\
\qquad\qquad okay \wedge \neg\, wait' \wedge (\mathbf{CSP1}(okay' \wedge do_{\mathcal{C}}(c, Sync) \wedge v' = v);\ (A)^t)\} & \\
 & [\text{Lemma J.5}] \\
= \{tr' - tr \mid okay \wedge ((okay' \wedge do_{\mathcal{C}}(c, Sync) \wedge v' = v);\ (A)^t)\} & [do_{\mathcal{C}}] \\
\quad \cup \{(tr' - tr) \frown \langle \checkmark \rangle \mid & \\
\qquad\qquad okay \wedge \neg\, wait' \wedge ((okay' \wedge do_{\mathcal{C}}(c, Sync) \wedge v' = v);\ (A)^t)\} &
\end{array}
$$

$$
= \left\{ \begin{array}{l} tr' - tr \\ \mid okay \wedge \left(\begin{array}{l} \left(\begin{array}{l} okay' \wedge v' = v \\ \wedge \left(\begin{array}{l} tr' = tr \wedge (c, Sync) \notin ref' \\ \lhd wait' \rhd \\ tr' = tr \frown \langle (c, Sync) \rangle \end{array} \right) \end{array} \right); \\ (A)^t \end{array} \right) \end{array} \right\} \qquad [\text{PC}]
$$

$$
\cup \left\{ \begin{array}{l} (tr' - tr) \frown \langle \checkmark \rangle \\ \mid okay \wedge \neg\, wait' \wedge \left(\begin{array}{l} \left(\begin{array}{l} okay' \wedge v' = v \\ \wedge \left(\begin{array}{l} tr' = tr \wedge (c, Sync) \notin ref' \\ \lhd wait' \rhd \\ tr' = tr \frown \langle (c, Sync) \rangle \end{array} \right) \end{array} \right); \\ (A)^t \end{array} \right) \end{array} \right\}
$$

$$= \left\{ \begin{array}{l} tr' - tr \\ \mid \left(okay \land \left(\begin{array}{l} \left(\begin{array}{l} okay' \land v' = v \land wait' \\ \land\, tr' = tr \land (c, Sync) \notin ref' \end{array} \right); \\ (A)^t \end{array} \right) \right) \\ \quad \lor \left(okay \land \left(\begin{array}{l} \left(\begin{array}{l} okay' \land v' = v \land \neg\, wait' \\ \land\, tr' = tr \frown \langle\, (c, Sync) \rangle \end{array} \right); \\ (A)^t \end{array} \right) \right) \end{array} \right\}$$

$$\cup \left\{ \begin{array}{l} (tr' - tr) \frown \langle \checkmark \rangle \\ \mid \left(okay \land \neg\, wait' \land \left(\begin{array}{l} \left(\begin{array}{l} okay' \land v' = v \land wait' \\ \land\, tr' = tr \land (c, Sync) \notin ref' \end{array} \right); \\ (A)^t \end{array} \right) \right) \\ \quad \lor \left(okay \land \neg\, wait' \land \left(\begin{array}{l} \left(\begin{array}{l} okay' \land v' = v \land \neg\, wait' \\ \land\, tr' = tr \frown \langle\, (c, Sync) \rangle \end{array} \right); \\ (A)^t \end{array} \right) \right) \end{array} \right\}$$

[Lemma J.25 (proviso)]

$$= \left\{ \begin{array}{l} tr' - tr \\ \mid (okay \land v' = v \land wait' \land tr' = tr \land (c, Sync) \notin ref') \\ \quad \lor \left(okay \land \left(\begin{array}{l} \left(\begin{array}{l} okay' \land v' = v \land \neg\, wait' \\ \land\, tr' = tr \frown \langle\, (c, Sync) \rangle \end{array} \right); \\ (A)^t \end{array} \right) \right) \end{array} \right\} \text{[PC and SC]}$$

$$\cup \left\{ \begin{array}{l} (tr' - tr) \frown \langle \checkmark \rangle \\ \mid (okay \land \neg\, wait' \land v' = v \land wait' \land tr' = tr \land (c, Sync) \notin ref') \\ \quad \lor \left(okay \land \neg\, wait' \land \left(\begin{array}{l} \left(\begin{array}{l} okay' \land v' = v \land \neg\, wait' \\ \land\, tr' = tr \frown \langle\, (c, Sync) \rangle \end{array} \right); \\ (A)^t \end{array} \right) \right) \end{array} \right\}$$

$$= \{ tr' - tr \mid okay \land v' = v \land wait' \land tr' = tr \land (c, Sync) \notin ref' \text{[Lemma J.16]}$$

$$\cup \left\{ \begin{array}{l} tr' - tr \\ \mid okay \land \left(\begin{array}{l} \left(\begin{array}{l} okay' \land v' = v \land \neg\, wait' \\ \land\, tr' = tr \frown \langle\, (c, Sync) \rangle \end{array} \right); \\ (A)^t \end{array} \right) \end{array} \right\}$$

$$\cup \left\{ \begin{array}{l} (tr' - tr) \frown \langle \checkmark \rangle \\ \mid okay \land \neg\, wait' \land \left(\begin{array}{l} \left(\begin{array}{l} okay' \land v' = v \land \neg\, wait' \\ \land\, tr' = tr \frown \langle\, (c, Sync) \rangle \end{array} \right); \\ (A)^t \end{array} \right) \end{array} \right\}$$

$$= \{ tr' - tr \mid okay \land v' = v \land wait' \land tr' = tr \land (c, Sync) \notin ref' \}$$
$$\cup \{ tr' - tr \mid okay \land (A)^t_f[tr \frown \langle (c, Sync) \rangle / tr] \}$$
$$\cup \{ (tr' - tr) \frown \langle \checkmark \rangle \mid okay \land \neg\, wait' \land (A)^t_f[tr \frown \langle (c, Sync) \rangle / tr] \}$$

[*Circus* Events $(c \equiv (c, Sync))$]

$$
\begin{aligned}
&= \{tr' - tr \mid okay \wedge v' = v \wedge wait' \wedge tr' = tr \wedge c \notin ref'\} && \text{[Lemma J.20 (proviso)]}\\
&\quad \cup \{tr' - tr \mid okay \wedge (A)^t_f[tr \frown \langle c\rangle / tr]\}\\
&\quad \cup \{(tr' - tr) \frown \langle \checkmark \rangle \mid okay \wedge \neg\, wait' \wedge (A)^t_f[tr \frown \langle c\rangle / tr]\}\\
&= \{tr' - tr \mid okay \wedge v' = v \wedge wait' \wedge tr' = tr \wedge c \notin ref'\} && \text{[SS, } - \text{ and cases]}\\
&\quad \cup \{\langle c\rangle \frown (tr' - tr) \mid okay \wedge (A)^t_f\}\\
&\quad \cup \{\langle c\rangle \frown ((tr' - tr) \frown \langle \checkmark \rangle) \mid okay \wedge \neg\, wait' \wedge (A)^t_f\}\\
&= \{\langle\rangle\} \cup \{\} && \text{[ST]}\\
&\quad \cup \{\langle c\rangle \frown (tr' - tr) \mid okay \wedge (A)^t_f\}\\
&\quad \cup \{\langle c\rangle \frown ((tr' - tr) \frown \langle \checkmark \rangle) \mid okay \wedge \neg\, wait' \wedge (A)^t_f\}\\
&= \{\langle\rangle\} && \text{[PC]}\\
&\quad \cup \{\langle c\rangle \frown (tr' - tr) \mid okay \wedge (A)^t_f\}\\
&\quad \cup \{\langle c\rangle \frown ((tr' - tr) \frown \langle \checkmark \rangle) \mid okay \wedge \neg\, wait' \wedge (A)^t_f\}\\
&= \{\langle\rangle\} && \text{[SC]}\\
&\quad \cup \{\langle c\rangle \frown (tr' - tr) \mid okay \wedge \neg\, wait \wedge okay' \wedge (A)\}\\
&\quad \cup \{\langle c\rangle \frown (tr' - tr) \frown \langle \checkmark \rangle \mid okay \wedge \neg\, wait \wedge okay' \wedge \neg\, wait' \wedge (A)\}\\
&= \{\langle\rangle\} && [A^n]\\
&\quad \cup \{\langle c\rangle \frown s \mid s \in \{tr' - tr \mid okay \wedge \neg\, wait \wedge okay' \wedge A)\}\}\\
&\quad \cup \{\langle c\rangle \frown s \mid s \in \{(tr' - tr) \frown \langle \checkmark \rangle \mid okay \wedge \neg\, wait \wedge okay' \wedge \neg\, wait' \wedge A\}\}\\
&= \{\langle\rangle\} && [A^t]\\
&\quad \cup \{\langle c\rangle \frown s \mid s \in \{tr' - tr \mid (A)^n\}\}\\
&\quad \cup \{\langle c\rangle \frown s \mid s \in \{(tr' - tr) \frown \langle \checkmark \rangle \mid \neg\, wait' \wedge (A)^n\}\}\\
&= \{\langle\rangle\} && \text{[SC and ST]}\\
&\quad \cup \{\langle c\rangle \frown s \mid s \in \{tr' - tr \mid (A)^n\}\}\\
&\quad \cup \{\langle c\rangle \frown s \mid s \in \{(tr' - tr) \frown \langle \checkmark \rangle \mid (A)^t\}\}\\
&= \{\langle\rangle\} && [traces^{\mathcal{UTP}}]\\
&\quad \cup \left\{ \langle c\rangle \frown s \mid s \in \left\{ \begin{array}{l} \{tr' - tr \mid (A)^n\} \\ \cup \{(tr' - tr) \frown \langle \checkmark \rangle \mid (A)^t\} \end{array} \right\} \right\}\\
&= \{\langle\rangle\} \cup \{\langle c\rangle \frown s \mid s \in traces^{\mathcal{UTP}}(A)\} && \text{[IH]}\\
&= \{\langle\rangle\} \cup \{\langle c\rangle \frown s \mid s \in traces(\Upsilon(A))\} && [traces]\\
&= traces(\mathsf{c} \rightarrow \Upsilon(A)) && [\Upsilon]\\
&= traces(\Upsilon(c \rightarrow A))
\end{aligned}
$$

Theorem J.6 $traces^{\mathcal{UTP}}(c.v \rightarrow A) = traces(\Upsilon(c.v \rightarrow A))$
provided A is $\boldsymbol{R}$

Inductive Hypothesis:

$$traces^{\mathcal{UTP}}(A) = traces(\Upsilon(A))$$

Proof. Identical to that of Theorem J.5, but replacing *Sync* by v.

Theorem J.7 $traces^{\mathcal{UTP}}(c!v \rightarrow A) = traces(\Upsilon(c.v \rightarrow A))$
provided A *is* $\boldsymbol{R}$

Proof. Using the *Circus* semantics of $c!v \rightarrow A \equiv c.v \rightarrow A$ and Theorem J.6.

Theorem J.8

$$traces^{\mathcal{UTP}}(c?x : P \rightarrow A) = traces(\Upsilon(c?x : P \rightarrow A))$$

provided

1. $c?x : P \rightarrow A$ *is* $\boldsymbol{R}$
2. $c?x : P \rightarrow A$ *is divergence-free*

Inductive Hypothesis (A):

$$\forall v : S \bullet traces^{\mathcal{UTP}}(A[v/x])) = traces(\Upsilon(A)[v/x])$$

Proof.

$$\begin{aligned}
& traces^{\mathcal{UTP}}(c?x : P \rightarrow A) && \text{[Property of \textit{Circus} input]}\\
& = traces^{\mathcal{UTP}}(\Box\, v : \{x : \delta(c) \mid P\} \bullet c.v \rightarrow A[v/x]) \\
& && \text{[Theorems J.11 and J.6 (IH)]}\\
& = traces(\Upsilon(\Box\, v : \{x : \delta(c) \mid P\} \bullet c.v \rightarrow A[v/x])) \\
& && \text{[Property of \textit{Circus} input]}\\
& = traces(\Upsilon(c?x : P \rightarrow A))
\end{aligned}$$

Theorem J.9

$$traces^{\mathcal{UTP}}(c?x \rightarrow A) = traces(\Upsilon(c?x \rightarrow A))$$

provided

1. *$c?x : P \rightarrow A$ is $\boldsymbol{R}$*
2. *$c?x : P \rightarrow A$ is divergence-free*
3. *$\forall v : S \bullet traces^{\mathcal{UTP}}(A[v/x])) = traces(\Upsilon(A)[v/x])$*

Proof.

Proof. Using the *Circus* semantics of $c?x \rightarrow A \equiv c?x : true \rightarrow A$ and Theorem J.8.

Theorem J.10 *$traces^{\mathcal{UTP}}(A \Box B) = traces(\Upsilon(A \Box B))$*
provided

1. *A and B are $\boldsymbol{R}$*
2. *A and B are divergence-free*

Inductive Hypothesis:

$$traces^{\mathcal{UTP}}(A) = traces(\Upsilon(A))$$
$$traces^{\mathcal{UTP}}(B) = traces(\Upsilon(B))$$

Proof.

$$\begin{array}{ll}
traces^{\mathcal{UTP}}(A \Box B) & [traces^{\mathcal{UTP}}] \\
= \{tr' - tr \mid (A \Box B)^n\} & [A^t] \\
\quad \cup \{(tr' - tr) \frown \langle \checkmark \rangle \mid (A \Box B)^t\} & \\
= \{tr' - tr \mid (A \Box B)^n\} & [A^n] \\
\quad \cup \{(tr' - tr) \frown \langle \checkmark \rangle \mid \neg\, wait' \wedge (A \Box B)^n\} & \\
= \left\{ \begin{array}{l} tr' - tr \\ \mid okay \wedge \neg\, wait \wedge okay' \wedge A \Box B \end{array} \right\} & [\text{PC}] \\
\quad \cup & \\
\quad \left\{ \begin{array}{l} (tr' - tr) \frown \langle \checkmark \rangle \\ \mid okay \wedge \neg\, wait \wedge okay' \wedge \neg\, wait' \wedge A \Box B \end{array} \right\} & \\
= \left\{ \begin{array}{l} tr' - tr \\ \mid okay \wedge (A \Box B)^t_f \end{array} \right\} & [\text{Lemma J.17}] \\
\quad \cup & \\
\quad \left\{ \begin{array}{l} (tr' - tr) \frown \langle \checkmark \rangle \\ \mid okay \wedge \neg\, wait' \wedge (A \Box B)^t_f \end{array} \right\} &
\end{array}$$

$$= \left\{ \begin{array}{l} tr' - tr \\ \mid okay \wedge \mathbf{CSP1} \left(\begin{array}{l} (\neg (A)^f_f \wedge \neg (B)^f_f) \\ \Rightarrow \\ \left(\begin{array}{l} ((A)^t_f \wedge (B)^t_f) \\ \lhd tr' = tr \wedge wait' \rhd \\ ((A)^t_f \vee (B)^t_f) \end{array} \right) \end{array} \right) \end{array} \right\}$$
$$\cup$$
$$\left\{ \begin{array}{l} (tr' - tr) \frown \langle \checkmark \rangle \\ \mid okay \wedge \neg wait' \wedge \mathbf{CSP1} \left(\begin{array}{l} (\neg (A)^f_f \wedge \neg (B)^f_f) \\ \Rightarrow \\ \left(\begin{array}{l} ((A)^t_f \wedge (B)^t_f) \\ \lhd tr' = tr \wedge wait' \rhd \\ ((A)^t_f \vee (B)^t_f) \end{array} \right) \end{array} \right) \end{array} \right\}$$

[Lemma J.4]

$$= \left\{ \begin{array}{l} tr' - tr \\ \mid okay \wedge \left(\begin{array}{l} (\neg (A)^f_f \wedge \neg (B)^f_f) \\ \Rightarrow \\ \left(\begin{array}{l} ((A)^t_f \wedge (B)^t_f) \\ \lhd tr' = tr \wedge wait' \rhd \\ ((A)^t_f \vee (B)^t_f) \end{array} \right) \end{array} \right) \end{array} \right\}$$
$$\cup$$
$$\left\{ \begin{array}{l} (tr' - tr) \frown \langle \checkmark \rangle \\ \mid okay \wedge \neg wait' \wedge \left(\begin{array}{l} (\neg (A)^f_f \wedge \neg (B)^f_f) \\ \Rightarrow \\ \left(\begin{array}{l} ((A)^t_f \wedge (B)^t_f) \\ \lhd tr' = tr \wedge wait' \rhd \\ ((A)^t_f \vee (B)^t_f) \end{array} \right) \end{array} \right) \end{array} \right\}$$

[Proviso 2 and PC]

$$= \left\{ \begin{array}{l} tr' - tr \\ \mid okay \wedge \left(\begin{array}{l} (tr' = tr \wedge wait' \wedge (A)^t_f \wedge (B)^t_f) \\ \vee (\neg (tr' = tr \wedge wait') \wedge (A)^t_f) \\ \vee (\neg (tr' = tr \wedge wait') \wedge (B)^t_f) \end{array} \right) \end{array} \right\} \quad \text{[SC]}$$
$$\cup$$
$$\{(tr' - tr) \frown \langle \checkmark \rangle \mid okay \wedge \neg wait' \wedge ((A)^t_f \vee (B)^t_f)\}$$

$$
\begin{aligned}
&= \{tr' - tr \mid okay \land tr' = tr \land wait' \land (A)^t_f \land (B)^t_f\} \\
&\quad \cup \{tr' - tr \mid okay \land \neg\, (tr' = tr \land wait') \land (A)^t_f\} \\
&\quad \cup \{tr' - tr \mid okay \land \neg\, (tr' = tr \land wait') \land (B)^t_f\} \\
&\quad \cup \{(tr' - tr) \frown \langle \checkmark \rangle \mid okay \land \neg\, wait' \land (A)^t_f\} \\
&\quad \cup \{(tr' - tr) \frown \langle \checkmark \rangle \mid okay \land \neg\, wait' \land (B)^t_f\} \\
&& [\text{Cases, } -\text{, PC and SC}] \\
&= \{\langle\rangle\} && [\text{Lemma J.21}] \\
&\quad \cup \{tr' - tr \mid okay \land \neg\, (tr' = tr \land wait') \land (A)^t_f\} \\
&\quad \cup \{tr' - tr \mid okay \land \neg\, (tr' = tr \land wait') \land (B)^t_f\} \\
&\quad \cup \{(tr' - tr) \frown \langle \checkmark \rangle \mid okay \land \neg\, wait' \land (A)^t_f\} \\
&\quad \cup \{(tr' - tr) \frown \langle \checkmark \rangle \mid okay \land \neg\, wait' \land (B)^t_f\} \\
&= \{\langle\rangle\} && [\text{PC}] \\
&\quad \cup \{tr' - tr \mid okay \land (A)^t_f\} \\
&\quad \cup \{(tr' - tr) \frown \langle \checkmark \rangle \mid okay \land \neg\, wait' \land (A)^t_f\} \\
&\quad \cup \{tr' - tr \mid okay \land (B)^t_f\} \\
&\quad \cup \{(tr' - tr) \frown \langle \checkmark \rangle \mid okay \land \neg\, wait' \land (A)^t_f\} \\
&= \{\langle\rangle\} && [A^n] \\
&\quad \cup \{tr' - tr \mid okay \land \neg\, wait \land okay' \land A\} \\
&\quad \cup \{(tr' - tr) \frown \langle \checkmark \rangle \mid okay \land \neg\, wait \land okay' \land \neg\, wait' \land A\} \\
&\quad \cup \{tr' - tr \mid okay \land \neg\, wait \land okay' \land B\} \\
&\quad \cup \{(tr' - tr) \frown \langle \checkmark \rangle \mid okay \land \neg\, wait \land okay' \land \neg\, wait' \land (B\} \\
&= \{\langle\rangle\} && [A^t] \\
&\quad \cup \{tr' - tr \mid (A)^n\} \\
&\quad \cup \{(tr' - tr) \frown \langle \checkmark \rangle \mid \neg\, wait' \land (A)^n\} \\
&\quad \cup \{tr' - tr \mid (B)^n\} \\
&\quad \cup \{(tr' - tr) \frown \langle \checkmark \rangle \mid \neg\, wait' \land (B)^n\} \\
&= \{\langle\rangle\} && [traces^{\mathcal{UTP}}] \\
&\quad \cup \{tr' - tr \mid (A)^n\} \\
&\quad \cup \{(tr' - tr) \frown \langle \checkmark \rangle \mid (A)^t\} \\
&\quad \cup \{tr' - tr \mid (B)^n\} \\
&\quad \cup \{(tr' - tr) \frown \langle \checkmark \rangle \mid (B)^t\} \\
&= \{\langle\rangle\} \cup traces^{\mathcal{UTP}}(A) \cup traces^{\mathcal{UTP}}(B) && [\text{IH}] \\
&= \{\langle\rangle\} \cup traces(\Upsilon(A)) \cup traces(\Upsilon(B)) && [traces] \\
&= \{\langle\rangle\} \cup traces(\Upsilon(A) \mathbin{\Box} \Upsilon(B)) && [traces \text{ prefix-closed}] \\
&= traces(\Upsilon(A) \mathbin{\Box} \Upsilon(B)) && [\Upsilon] \\
&= traces(\Upsilon(A \mathbin{\Box} B))
\end{aligned}
$$

Theorem J.11

$$\mathit{traces}^{\mathcal{UTP}}(\Box\, x : S \bullet A) = \mathit{traces}(\Upsilon(\Box\, x : S \bullet A))$$

provided

1. $\forall\, i : S \bullet A[v_i/x]$ *is* $\boldsymbol{R}$
2. $\forall\, i : S \bullet A[v_i/x]$ *is divergence-free*

Inductive Hypothesis (A):

$$\forall\, i : S \bullet \mathit{traces}^{\mathcal{UTP}}(A[v_i/x]) = \mathit{traces}(\Upsilon(A)[v_i/x])$$

Proof. By induction on S

Base Case. $S = \{\}$
Proof.

$$\begin{aligned}
&\mathit{traces}^{\mathcal{UTP}}(\Box\, x : S \bullet A) && [\text{Assumption}]\\
&= \mathit{traces}^{\mathcal{UTP}}(\Box\, x : \{\} \bullet A) && [\text{Property of } \Box]\\
&= \mathit{traces}^{\mathcal{UTP}}(\mathit{Stop}) && [\text{Theorem J.3}]\\
&= \mathit{traces}(\Upsilon(\mathit{Stop})) && [\text{Property of } \Box]\\
&= \mathit{traces}(\Upsilon(\Box\, x : \{\} \bullet A)) && [\text{Assumption}]\\
&= \mathit{traces}(\Upsilon(\Box\, x : S \bullet A))
\end{aligned}$$

Inductive Hypothesis (S):

$$\mathit{traces}^{\mathcal{UTP}}(\Box\, x : S \bullet A) = \mathit{traces}(\Upsilon(\Box\, x : S \bullet A))$$

Inductive Step

$$\mathit{traces}^{\mathcal{UTP}}(\Box\, x : S \cup \{v_i\} \bullet A) = \mathit{traces}(\Upsilon(\Box\, x : S \cup \{v_i\} \bullet A))$$

Proof.

$$\begin{aligned}
&\mathit{traces}^{\mathcal{UTP}}(\Box\, x : S \cup \{v_i\} \bullet A) && [\Box]\\
&= \mathit{traces}^{\mathcal{UTP}}(A[v_i/x] \Box (\Box\, x : S \setminus \{v_i\} \bullet A))\\
& && [\text{Theorem J.10 (Provisos, IH-}A\text{ and IH-}S)]\\
&= \mathit{traces}(\Upsilon(A[v_i/x] \Box (\Box\, x : S \setminus \{v_i\} \bullet A))) && [\Box]\\
&= \mathit{traces}(\Upsilon(\Box\, x : S \cup \{v_i\} \bullet A))
\end{aligned}$$

Theorem J.12 $traces^{\mathcal{UTP}}(A \sqcap B) = traces(\Upsilon(A \sqcap B))$

Inductive Hypothesis:

$$traces^{\mathcal{UTP}}(A) = traces(\Upsilon(A))$$
$$traces^{\mathcal{UTP}}(B) = traces(\Upsilon(B))$$

Proof.

$$\begin{array}{lr}
traces^{\mathcal{UTP}}(A \sqcap B) & [traces^{\mathcal{UTP}}] \\
= \{tr' - tr \mid (A \sqcap B)^n\} & [A^t] \\
\quad \cup \{(tr' - tr) ^\frown \langle \checkmark \rangle \mid (A \sqcap B)^t\} & \\
= \{tr' - tr \mid (A \sqcap B)^n\} & [A^n] \\
\quad \cup \{(tr' - tr) ^\frown \langle \checkmark \rangle \mid \neg\, wait' \wedge (A \sqcap B)^n\} & \\
= \left\{ \begin{array}{l} tr' - tr \\ \mid okay \wedge \neg\, wait \wedge okay' \wedge A \sqcap B \end{array} \right\} & [\text{PC}] \\
\quad \cup & \\
\quad \left\{ \begin{array}{l} (tr' - tr) ^\frown \langle \checkmark \rangle \\ \mid okay \wedge \neg\, wait \wedge okay' \wedge \neg\, wait' \wedge A \sqcap B \end{array} \right\} & \\
= \left\{ \begin{array}{l} tr' - tr \\ \mid okay \wedge (A \sqcap B)^t_f \end{array} \right\} & [\ ] \\
\quad \cup & \\
\quad \left\{ \begin{array}{l} (tr' - tr) ^\frown \langle \checkmark \rangle \\ \mid okay \wedge \neg\, wait' \wedge (A \sqcap B)^t_f \end{array} \right\} & \\
= \left\{ \begin{array}{l} tr' - tr \\ \mid okay \wedge (A \vee B)^t_f \end{array} \right\} & [\text{PC}] \\
\quad \cup & \\
\quad \left\{ \begin{array}{l} (tr' - tr) ^\frown \langle \checkmark \rangle \\ \mid okay \wedge \neg\, wait' \wedge (A \vee B)^t_f \end{array} \right\} & \\
= \left\{ \begin{array}{l} tr' - tr \\ \mid okay \wedge (A)^t_f \vee (B)^t_f \end{array} \right\} & [\text{PC, SC and ST}] \\
\quad \cup & \\
\quad \left\{ \begin{array}{l} (tr' - tr) ^\frown \langle \checkmark \rangle \\ \mid okay \wedge \neg\, wait' \wedge (A)^t_f \vee (B)^t_f \end{array} \right\} & \\
= \{tr' - tr \mid okay \wedge (A)^t_f\} & [\text{PC}] \\
\quad \cup \{(tr' - tr) ^\frown \langle \checkmark \rangle \mid okay \wedge \neg\, wait' \wedge (A)^t_f\} & \\
\quad \cup \{tr' - tr \mid okay \wedge (B)^t_f\} & \\
\quad \cup \{(tr' - tr) ^\frown \langle \checkmark \rangle \mid okay \wedge \neg\, wait' \wedge (A)^t_f\} &
\end{array}$$

$$
\begin{aligned}
&= \{tr' - tr \mid okay \land \neg\, wait \land okay' \land A\} && [A^n]\\
&\quad \cup \{(tr' - tr) \frown \langle \checkmark \rangle \mid okay \land \neg\, wait \land okay' \land \neg\, wait' \land A\}\\
&\quad \cup \{tr' - tr \mid okay \land \neg\, wait \land okay' \land B\}\\
&\quad \cup \{(tr' - tr) \frown \langle \checkmark \rangle \mid okay \land \neg\, wait \land okay' \land \neg\, wait' \land (B\}\\
&= \{tr' - tr \mid (A)^n\} && [A^t]\\
&\quad \cup \{(tr' - tr) \frown \langle \checkmark \rangle \mid \neg\, wait' \land (A)^n\}\\
&\quad \cup \{tr' - tr \mid (B)^n\}\\
&\quad \cup \{(tr' - tr) \frown \langle \checkmark \rangle \mid \neg\, wait' \land (B)^n\}\\
&= \{tr' - tr \mid (A)^n\} && [traces^{\mathcal{UTP}}]\\
&\quad \cup \{(tr' - tr) \frown \langle \checkmark \rangle \mid (A)^t\}\\
&\quad \cup \{tr' - tr \mid (B)^n\}\\
&\quad \cup \{(tr' - tr) \frown \langle \checkmark \rangle \mid (B)^t\}\\
&= traces^{\mathcal{UTP}}(A) \cup traces^{\mathcal{UTP}}(B) && [\text{IH}]\\
&= traces(\Upsilon(A)) \cup traces(\Upsilon(A)) && [traces]\\
&= traces(\Upsilon(A) \sqcap \Upsilon(B)) && [\Upsilon]\\
&= traces(\Upsilon(A \sqcap B))
\end{aligned}
$$

Theorem J.13

$$traces^{\mathcal{UTP}}(\textstyle\bigsqcap x : S \bullet A) = traces(\Upsilon(\textstyle\bigsqcap x : S \bullet A))$$

provided

1. $\forall\, i : S \bullet A[v_i/x]$ *is* $\boldsymbol{R}$
2. $\forall\, i : S \bullet A[v_i/x]$ *is divergence-free*
3. $S \neq \{\}$

Inductive Hypothesis (A):

$$\forall\, i : S \bullet traces^{\mathcal{UTP}}(A[v_i/x]) = traces(\Upsilon(A)[v_i/x])$$

Proof. By induction on S

Base Case. $S = \{v\}$
Proof.

$$traces^{\mathcal{UTP}}(\textstyle\bigsqcap x : S \bullet A) \qquad [\text{Assumption}]$$

$= traces^{\mathcal{UTP}}(\bigsqcap x : \{v\} \bullet A)$ [$\bigsqcap$]
$= traces^{\mathcal{UTP}}(A[v/x])$ [IH]
$= traces(\Upsilon(A[v/x]))$ [$\bigsqcap$]
$= traces(\Upsilon(\bigsqcap x : \{v\} \bullet A))$ [Assumption]
$= traces(\Upsilon(\bigsqcap x : S \bullet A))$

Inductive Hypothesis (S):

$traces^{\mathcal{UTP}}(\bigsqcap x : S \bullet A) = traces(\Upsilon(\bigsqcap x : S \bullet A))$

Inductive Step

$traces^{\mathcal{UTP}}(\bigsqcap x : S \cup \{v_i\} \bullet A) = traces(\Upsilon(\bigsqcap x : S \cup \{v_i\} \bullet A))$

Proof.

$traces^{\mathcal{UTP}}(\bigsqcap x : S \cup \{v_i\} \bullet A)$ [$\bigsqcap$]
$= traces^{\mathcal{UTP}}(A[v_i/x] \sqcap (\bigsqcap x : S \setminus \{v_i\} \bullet A))$
[Theorem J.12 (Provisos, IH-A and IH-S)]
$= traces(\Upsilon(A[v_i/x] \Box (\bigsqcap x : S \setminus \{v_i\} \bullet A)))$ [$\bigsqcap$]
$= traces(\Upsilon(\bigsqcap x : S \cup \{v_i\} \bullet A))$

Theorem J.14 $traces^{\mathcal{UTP}}(g \ \& \ A) = traces(\Upsilon(g \ \& \ A))$

Inductive Hypothesis:

$traces^{\mathcal{UTP}}(A) = traces(\Upsilon(A))$

Proof. The proof will be conducted by cases on g.

Case 1. g is *false*

Proof.

$traces^{\mathcal{UTP}}(g \ \& \ A)$ [Assumption]
$= traces^{\mathcal{UTP}}(false \ \& \ A)$ [Law 38]
$= traces^{\mathcal{UTP}}(Stop)$ [Theorem J.3]
$= traces(\Upsilon(Stop))$ [Law 38]
$= traces(\Upsilon(false \ \& \ A))$ [Assumption]
$= traces(\Upsilon(g \ \& \ A))$

Case 2. g is *true*
Proof.

$$
\begin{aligned}
& \mathit{traces}^{\mathcal{UTP}}(g \,\&\, A) && \text{[Assumption]}\\
&= \mathit{traces}^{\mathcal{UTP}}(\mathit{true} \,\&\, A) && \text{[Law 37]}\\
&= \mathit{traces}^{\mathcal{UTP}}(A) && \text{[IH]}\\
&= \mathit{traces}(\Upsilon(A)) && \text{[Law 37]}\\
&= \mathit{traces}(\Upsilon(\mathit{true} \,\&\, A)) && \text{[Assumption]}\\
&= \mathit{traces}(\Upsilon(g \,\&\, A))
\end{aligned}
$$

Theorem J.15 $\mathit{traces}^{\mathcal{UTP}}(P;\ Q) = \mathit{traces}(\Upsilon(P;\ Q))$
provided

1. *P and Q are divergence-free*
2. *$P = \boldsymbol{R}(P_{pre} \vdash P_{post})$ and $Q = \boldsymbol{R}(Q_{pre} \vdash Q_{post})$*
3. *P_{pre} does not mention any dashed variable*
4. *P_{post} and Q_{post} are **R1** and **R2***

Inductive Hypothesis:

$\mathit{traces}^{\mathcal{UTP}}(P) = \mathit{traces}(\Upsilon(P))$
and
$\mathit{traces}^{\mathcal{UTP}}(Q) = \mathit{traces}(\Upsilon(Q))$

Proof.

$$
\begin{aligned}
& \mathit{traces}^{\mathcal{UTP}}(P;\ Q) && [\mathit{traces}^{\mathcal{UTP}}]\\
&= \{tr' - tr \mid (P;\ Q)^n\} && [A^t]\\
&\quad \cup \{(tr' - tr) ^\frown \langle \checkmark \rangle \mid (P;\ Q)^t\}\\
&= \{tr' - tr \mid (P;\ Q)^n\} && [A^n]\\
&\quad \cup \{(tr' - tr) ^\frown \langle \checkmark \rangle \mid \neg\, wait' \wedge (P;\ Q)^n\}\\
&= \{tr' - tr \mid okay \wedge \neg\, wait \wedge okay' \wedge P;\ Q\} && \text{[PC]}\\
&\quad \cup \{(tr' - tr) ^\frown \langle \checkmark \rangle \mid okay \wedge \neg\, wait \wedge okay' \wedge \neg\, wait' \wedge P;\ Q\}\\
&= \{tr' - tr \mid okay \wedge (P;\ Q)^t_f\} && \text{[Lemma J.19 (Assumptions)]}\\
&\quad \cup \{(tr' - tr) ^\frown \langle \checkmark \rangle \mid okay \wedge \neg\, wait' \wedge (P;\ Q)^t_f\}
\end{aligned}
$$

$$
\begin{array}{l}
= \left\{ \begin{array}{l} tr' - tr \mid \\ \qquad okay \land \mathbf{CSP1} \left(\begin{array}{l} (wait' \land P_{post}) \\ \lor ((okay' \land \neg\, wait' \land P_{post});\ Q_{post}) \end{array} \right) \end{array} \right\} \\
\quad \cup \left\{ \begin{array}{l} (tr' - tr) \frown \langle \checkmark \rangle \mid \\ \qquad okay \land \neg\, wait' \land \mathbf{CSP1} \left(\begin{array}{l} (wait' \land P_{post}) \\ \lor ((okay' \land \neg\, wait' \land P_{post});\ Q_{post}) \end{array} \right) \end{array} \right\}
\end{array}
$$

[Lemma J.4, PC, SC and ST]

$$
\begin{array}{l}
= \{ tr' - tr \mid okay \land wait' \land P_{post} \} \\
\quad \cup \{ tr' - tr \mid okay \land ((okay' \land \neg\, wait' \land P_{post});\ Q_{post}) \} \\
\quad \cup \{ (tr' - tr) \frown \langle \checkmark \rangle \mid okay \land \neg\, wait' \land ((okay' \land \neg\, wait' \land P_{post});\ Q_{post}) \}
\end{array}
$$

[Sequence and PC]

$$
\begin{array}{l}
= \{ tr' - tr \mid okay \land wait' \land P_{post} \} \\
\quad \cup \{ tr' - tr \mid ((okay \land \neg\, wait' \land P_{post});\ (okay \land Q_{post})) \} \\
\quad \cup \{ (tr' - tr) \frown \langle \checkmark \rangle \mid ((okay \land \neg\, wait' \land P_{post});\ (okay \land \neg\, wait' \land Q_{post})) \}
\end{array}
$$

[Sequence, PC and SC]

[SC and ST'(t might be $\langle\rangle$ based on Lemma J.15)]

$$
\begin{array}{l}
= \{ tr' - tr \mid okay \land wait' \land P_{post} \} \\
\quad \cup \left\{ \begin{array}{l} s \frown t \mid \\ \qquad s \in \{ tr' - tr \mid okay \land \neg\, wait' \land P_{post} \} \\ \qquad \land\ t \in \{ tr' - tr \mid okay \land Q_{post} \} \end{array} \right\} \\
\quad \cup \left\{ \begin{array}{l} s \frown t \frown \langle \checkmark \rangle \mid \\ \qquad s \in \{ tr' - tr \mid okay \land \neg\, wait' \land P_{post} \} \\ \qquad \land\ t \in \{ tr' - tr \mid okay \land \neg\, wait' \land Q_{post} \} \end{array} \right\}
\end{array}
$$

[Cases on $wait'$, PC, SC and ST]

$$
\begin{array}{l}
= \{ tr' - tr \mid okay \land wait' \land P_{post} \} \\
\quad \cup \{ tr' - tr \mid okay \land \neg\, wait' \land P_{post} \} \\
\quad \cup \left\{ \begin{array}{l} s \frown t \mid \\ \qquad s \in \{ tr' - tr \mid okay \land \neg\, wait' \land P_{post} \} \\ \qquad \land\ t \in \{ tr' - tr \mid okay \land Q_{post} \} \end{array} \right\} \\
\quad \cup \left\{ \begin{array}{l} s \frown t \frown \langle \checkmark \rangle \mid \\ \qquad s \in \{ tr' - tr \mid okay \land \neg\, wait' \land P_{post} \} \\ \qquad \land\ t \in \{ tr' - tr \mid okay \land \neg\, wait' \land Q_{post} \} \end{array} \right\}
\end{array}
$$

[SC]

$$
\begin{array}{l}
= \{ tr' - tr \mid okay \land P_{post} \} \\
\quad \cup \left\{ \begin{array}{l} s \frown t \mid \\ \qquad s \in \{ tr' - tr \mid okay \land \neg\, wait' \land P_{post} \} \\ \qquad \land\ t \in \{ tr' - tr \mid okay \land Q_{post} \} \end{array} \right\} \\
\quad \cup \left\{ \begin{array}{l} s \frown t \frown \langle \checkmark \rangle \mid \\ \qquad s \in \{ tr' - tr \mid okay \land \neg\, wait' \land P_{post} \} \\ \qquad \land\ t \in \{ tr' - tr \mid okay \land \neg\, wait' \land Q_{post} \} \end{array} \right\}
\end{array}
$$

$$
\begin{array}{l}
= \{tr' - tr \mid okay \land P_{post}\} \\
\cup \left\{ \begin{array}{l} s ^\frown t \mid \\ \qquad s \in \{tr' - tr \mid okay \land \neg\, wait' \land P_{post}\} \\ \qquad \land\ t \in \{tr' - tr \mid okay \land Q_{post}\} \end{array} \right\} \\
\cup \left\{ \begin{array}{l} s ^\frown t \mid \\ \qquad s \in \{tr' - tr \mid okay \land \neg\, wait' \land P_{post}\} \\ \qquad \land\ t \in \{(tr' - tr) ^\frown \langle \checkmark \rangle \mid okay \land \neg\, wait' \land Q_{post}\} \end{array} \right\}
\end{array}
$$

[Lemma J.4 and Assumption 4]

$$
\begin{array}{l}
= \{tr' - tr \mid okay \land \mathbf{CSP1}(\mathbf{R1}(\mathbf{R2}(P_{post})))\} \\
\cup \left\{ \begin{array}{l} s ^\frown t \mid \\ \qquad s \in \{tr' - tr \mid okay \land \neg\, wait' \land \mathbf{CSP1}(\mathbf{R1}(\mathbf{R2}(P_{post})))\} \\ \qquad \land\ t \in \{tr' - tr \mid okay \land \mathbf{CSP1}(\mathbf{R1}(\mathbf{R2}(Q_{post})))\} \end{array} \right\} \\
\cup \left\{ \begin{array}{l} s ^\frown t \mid \\ \qquad s \in \{tr' - tr \mid okay \land \neg\, wait' \land \mathbf{CSP1}(\mathbf{R1}(\mathbf{R2}(P_{post})))\} \\ \qquad \land\ t \in \{(tr' - tr) ^\frown \langle \checkmark \rangle \mid okay \land \neg\, wait' \land \mathbf{CSP1}(\mathbf{R1}(\mathbf{R2}(Q_{post})))\} \end{array} \right\}
\end{array}
$$

[Lemma J.8 and PC]

$$
\begin{array}{l}
= \{tr' - tr \mid okay \land (\mathbf{R}(true \vdash P_{post}))^t_f\} \\
\cup \left\{ \begin{array}{l} s ^\frown t \mid \\ \qquad s \in \{tr' - tr \mid okay \land \neg\, wait' \land (\mathbf{R}(true \vdash P_{post}))^t_f\} \\ \qquad \land\ t \in \{tr' - tr \mid okay \land (\mathbf{R}(true \vdash Q_{post}))^t_f\} \end{array} \right\} \\
\cup \left\{ \begin{array}{l} s ^\frown t \mid \\ \qquad s \in \{tr' - tr \mid okay \land \neg\, wait' \land (\mathbf{R}(true \vdash P_{post}))^t_f\} \\ \qquad \land\ t \in \{(tr' - tr) ^\frown \langle \checkmark \rangle \mid okay \land \neg\, wait' \land (\mathbf{R}(true \vdash Q_{post}))^t_f\} \end{array} \right\}
\end{array}
$$

[Assumption 1]

$$
\begin{array}{l}
= \{tr' - tr \mid okay \land (\mathbf{R}(P_{pre} \vdash P_{post}))^t_f\} \\
\cup \left\{ \begin{array}{l} s ^\frown t \mid \\ \qquad s \in \{tr' - tr \mid okay \land \neg\, wait' \land (\mathbf{R}(P_{pre} \vdash P_{post}))^t_f\} \\ \qquad \land\ t \in \{tr' - tr \mid okay \land (\mathbf{R}(Q_{pre} \vdash Q_{post}))^t_f\} \end{array} \right\} \\
\cup \left\{ \begin{array}{l} s ^\frown t \mid \\ \qquad s \in \{tr' - tr \mid okay \land \neg\, wait' \land (\mathbf{R}(P_{pre} \vdash P_{post}))^t_f\} \\ \qquad \land\ t \in \{(tr' - tr) ^\frown \langle \checkmark \rangle \mid okay \land \neg\, wait' \land (\mathbf{R}(Q_{pre} \vdash Q_{post}))^t_f\} \end{array} \right\}
\end{array}
$$

[Assumption 2]

$$
\begin{array}{l}
= \{tr' - tr \mid okay \land (P)^t_f\} \\
\cup \left\{ \begin{array}{l} s \frown t \mid \\ \qquad s \in \{tr' - tr \mid okay \land \neg\, wait' \land (P)^t_f\} \\ \qquad \land\ t \in \{tr' - tr \mid okay \land (Q)^t_f\} \end{array} \right\} \\
\cup \left\{ \begin{array}{l} s \frown t \mid \\ \qquad s \in \{tr' - tr \mid okay \land \neg\, wait' \land (P)^t_f\} \\ \qquad \land\ t \in \{(tr' - tr) \frown \langle \checkmark \rangle \mid okay \land \neg\, wait' \land (Q)^t_f\} \end{array} \right\}
\end{array}
$$

[PC]

$$
\begin{array}{l}
= \{tr' - tr \mid okay \land \neg\, wait \land okay' \land (P)\} \\
\cup \left\{ \begin{array}{l} s \frown t \mid \\ \qquad s \in \{tr' - tr \mid okay \land \neg\, wait \land okay' \land \neg\, wait' \land (P)\} \\ \qquad \land\ t \in \{tr' - tr \mid okay \land \neg\, wait \land okay' \land (Q)\} \end{array} \right\} \\
\cup \left\{ \begin{array}{l} s \frown t \mid \\ \qquad s \in \{tr' - tr \mid okay \land \neg\, wait \land okay' \land \neg\, wait' \land (P)\} \\ \qquad \land\ t \in \{(tr' - tr) \frown \langle \checkmark \rangle \mid okay \land \neg\, wait \land okay' \land \neg\, wait' \land (Q)\} \end{array} \right\}
\end{array}
$$

$[A^n]$

$$
\begin{array}{l}
= \{tr' - tr \mid (P)^n\} \\
\cup \left\{ \begin{array}{l} s \frown t \mid \\ \qquad s \in \{tr' - tr \mid \neg\, wait' \land (P)^n\} \\ \qquad \land\ t \in \{tr' - tr \mid (Q)^n\} \end{array} \right\} \\
\cup \left\{ \begin{array}{l} s \frown t \mid \\ \qquad s \in \{tr' - tr \mid \neg\, wait' \land (P)^n\} \\ \qquad \land\ t \in \{(tr' - tr) \frown \langle \checkmark \rangle \mid \neg\, wait' \land (Q)^n\} \end{array} \right\}
\end{array}
$$

$[A^t]$

$$
\begin{array}{l}
= \{tr' - tr \mid (P)^n\} \\
\cup \left\{ \begin{array}{l} s \frown t \mid \\ \qquad s \in \{tr' - tr \mid (P)^t\} \\ \qquad \land\ t \in \{tr' - tr \mid (Q)^n\} \end{array} \right\} \\
\cup \left\{ \begin{array}{l} s \frown t \mid \\ \qquad s \in \{tr' - tr \mid (P)^t\} \\ \qquad \land\ t \in \{(tr' - tr) \frown \langle \checkmark \rangle \mid (Q)^t\} \end{array} \right\}
\end{array}
$$

[PC and SC $(tr, tr' : \operatorname{seq} \Sigma$ and $\checkmark \notin \Sigma)$]

$$
\begin{array}{l}
= \{tr' - tr \mid (P)^n\} \\
\cup \left\{ \begin{array}{l} s \frown t \mid \\ \qquad s \frown \langle \checkmark \rangle \in \{(tr' - tr) \frown \langle \checkmark \rangle \mid (P)^t\} \\ \qquad \land\ t \in \left(\begin{array}{l} \{tr' - tr \mid (Q)^n\} \\ \cup \{(tr' - tr) \frown \langle \checkmark \rangle \mid (Q)^t\} \end{array} \right) \end{array} \right\}
\end{array}
$$

[SC and ST $(tr, tr' : \operatorname{seq} \Sigma$ and $\checkmark \notin \Sigma)$]

$$
\begin{aligned}
&= \left(\begin{pmatrix} \{tr' - tr \mid (P)^n\} \\ \cup \{(tr' - tr) \frown \langle \checkmark \rangle \mid (P)^t\} \end{pmatrix} \cap \Sigma^* \right) \\
&\quad \cup \left\{ \begin{array}{l} s \frown t \mid \\ \qquad s \frown \langle \checkmark \rangle \in \begin{pmatrix} \{tr' - tr \mid (P)^n\} \\ \cup \{(tr' - tr) \frown \langle \checkmark \rangle \mid (P)^t\} \end{pmatrix} \\ \qquad \wedge\, t \in \begin{pmatrix} \{tr' - tr \mid (Q)^n\} \\ \cup \{(tr' - tr) \frown \langle \checkmark \rangle \mid (Q)^t\} \end{pmatrix} \end{array} \right\} && [\mathit{traces}^{\mathcal{UTP}}] \\
&= (\mathit{traces}^{\mathcal{UTP}}(P) \cap \Sigma^*) && [\mathrm{IH}] \\
&\quad \cup \{s \frown t \mid s \frown \langle \checkmark \rangle \in \mathit{traces}^{\mathcal{UTP}}(P) \wedge t \in \mathit{traces}^{\mathcal{UTP}}(Q)\} \\
&= (\mathit{traces}(\Upsilon(P)) \cap \Sigma^*) \cup \{s \frown t \mid s \frown \langle \checkmark \rangle \in \mathit{traces}(\Upsilon(P)) \wedge t \in \mathit{traces}(\Upsilon(Q))\} && [\mathit{traces}] \\
&= \mathit{traces}(\Upsilon(P);\ \Upsilon(Q)) && [\Upsilon] \\
&= \mathit{traces}(\Upsilon(P;\ Q))
\end{aligned}
$$

Theorem J.16

$$\mathit{traces}^{\mathcal{UTP}}(\fatsemi\ x : S \bullet A) = \mathit{traces}(\Upsilon(\fatsemi\ x : S \bullet A))$$

provided

1. $\forall\, i : S \bullet A[v_i/x]$ *is* $\boldsymbol{R}$
2. $\forall\, i : S \bullet A[v_i/x]$ *is divergence-free*

Inductive Hypothesis (A):

$$\forall\, i : S \bullet \mathit{traces}^{\mathcal{UTP}}(A[v_i/x]) = \mathit{traces}(\Upsilon(A)[v_i/x])$$

Proof. By induction on S

Base Case. $S = \langle\rangle$
Proof.

$$
\begin{aligned}
&\mathit{traces}^{\mathcal{UTP}}(\fatsemi\ x : S \bullet A) && [\text{Assumption}] \\
&= \mathit{traces}^{\mathcal{UTP}}(\fatsemi\ x : \langle\rangle \bullet A) && [\text{Property of } \fatsemi] \\
&= \mathit{traces}^{\mathcal{UTP}}(\mathit{Skip}) && [\text{Theorem J.2}] \\
&= \mathit{traces}(\Upsilon(\mathit{Skip})) && [\text{Property of } \fatsemi] \\
&= \mathit{traces}(\Upsilon(\fatsemi\ x : \langle\rangle \bullet A)) && [\text{Assumption}]
\end{aligned}
$$

$$= traces(\Upsilon(⨟\ x : S \bullet A))$$

Inductive Hypothesis (S):

$$traces^{\mathcal{UTP}}(⨟\ x : S \bullet A) = traces(\Upsilon(⨟\ x : S \bullet A))$$

Inductive Step

$$traces^{\mathcal{UTP}}(⨟\ x : S \cup \{v_i\} \bullet A) = traces(\Upsilon(⨟\ x : S \cup \{v_i\} \bullet A))$$

Proof.

$$\begin{aligned}
&traces^{\mathcal{UTP}}(⨟\ x : S \bullet A) && [⨟]\\
&= traces^{\mathcal{UTP}}(A[head(s)/x];\ (⨟\ x : tail(S) \bullet A))\\
& && [\text{Theorem J.15 (Provisos, IH-}A\text{ and IH-}S)]\\
&= traces(\Upsilon(A[head(v_i)/x];\ (⨟\ x : tail(S) \bullet A))) && [⨟]\\
&= traces(\Upsilon(⨟\ x : S \bullet A))
\end{aligned}$$

Theorem J.17

$$traces^{\mathcal{UTP}}(P \llbracket ns_1 \mid cs \mid ns_2 \rrbracket Q)$$
$$=$$
$$traces(\Upsilon(P \llbracket ns_1 \mid cs \mid ns_2 \rrbracket Q))$$

provided

1. *P and Q are divergence-free*

Inductive Hypothesis:

$traces^{\mathcal{UTP}}(P) = traces(\Upsilon(P))$
and
$traces^{\mathcal{UTP}}(Q) = traces(\Upsilon(Q))$

Proof.

$$\begin{aligned}
&traces^{\mathcal{UTP}}(P \llbracket ns_1 \mid cs \mid ns_2 \rrbracket Q) && [traces^{\mathcal{UTP}}]\\
&= \{tr' - tr \mid (P \llbracket ns_1 \mid cs \mid ns_2 \rrbracket Q)^n\} && [A^t]\\
&\quad \cup \{(tr' - tr) \frown \langle \checkmark \rangle \mid (P \llbracket ns_1 \mid cs \mid ns_2 \rrbracket Q)^t\}
\end{aligned}$$

$$
\begin{aligned}
&= \{tr' - tr \mid (P \llbracket ns_1 \mid cs \mid ns_2 \rrbracket Q)^n\} && [A^n]\\
&\quad \cup \{(tr' - tr) \frown \langle \checkmark \rangle \mid \neg\, wait' \land (P \llbracket ns_1 \mid cs \mid ns_2 \rrbracket Q)^n\}\\
&= \{tr' - tr \mid okay \land \neg\, wait \land okay' \land P \llbracket ns_1 \mid cs \mid ns_2 \rrbracket Q\} && [\text{PC}]\\
&\quad \cup \{(tr' - tr) \frown \langle \checkmark \rangle \mid okay \land \neg\, wait \land okay' \land \neg\, wait' \land P \llbracket ns_1 \mid cs \mid ns_2 \rrbracket Q\}\\
&= \{tr' - tr \mid okay \land (P \llbracket ns_1 \mid cs \mid ns_2 \rrbracket Q)^t_f\}\\
&\quad \cup \{(tr' - tr) \frown \langle \checkmark \rangle \mid okay \land \neg\, wait' \land (P \llbracket ns_1 \mid cs \mid ns_2 \rrbracket Q)^t_f\}\\
& && [\text{Lemma J.28 (Assumptions)}]
\end{aligned}
$$

$$
= \left\{
\begin{array}{l}
tr' - tr \mid \\
\qquad okay \\
\qquad \land\, \mathbf{CSP1}
\left(
\begin{array}{l}
\mathbf{R1}\left(
\begin{array}{l}
\exists\, 1.tr', 2.tr' \bullet (P^f_f;\ 1.tr' = tr)\\
\qquad \land (Q_f;\ 2.tr' = tr)\\
\qquad \land 1.tr' \upharpoonright cs = 2.tr' \upharpoonright cs
\end{array}
\right)\\
\lor\, \mathbf{R1}\left(
\begin{array}{l}
\exists\, 1.tr', 2.tr' \bullet (P_f;\ 1.tr' = tr)\\
\qquad \land (Q^f_f;\ 2.tr' = tr)\\
\qquad \land 1.tr' \upharpoonright cs = 2.tr' \upharpoonright cs
\end{array}
\right)\\
\lor \left(
\left(
\begin{array}{l}
(P^t_f;\ U1(out\alpha\, P))\\
\land (Q^t_f;\ U2(out\alpha\, Q))
\end{array}
\right)_{+\{v,tr\}} ;\ M_{\parallel_{cs}}
\right)
\end{array}
\right)
\end{array}
\right\}
$$

$$
\cup \left\{
\begin{array}{l}
(tr' - tr) \frown \langle \checkmark \rangle \mid \\
\qquad okay \land \neg\, wait' \\
\qquad \land\, \mathbf{CSP1}
\left(
\begin{array}{l}
\mathbf{R1}\left(
\begin{array}{l}
\exists\, 1.tr', 2.tr' \bullet (P^f_f;\ 1.tr' = tr)\\
\qquad \land (Q_f;\ 2.tr' = tr)\\
\qquad \land 1.tr' \upharpoonright cs = 2.tr' \upharpoonright cs
\end{array}
\right)\\
\lor\, \mathbf{R1}\left(
\begin{array}{l}
\exists\, 1.tr', 2.tr' \bullet (P_f;\ 1.tr' = tr)\\
\qquad \land (Q^f_f;\ 2.tr' = tr)\\
\qquad \land 1.tr' \upharpoonright cs = 2.tr' \upharpoonright cs
\end{array}
\right)\\
\lor \left(
\left(
\begin{array}{l}
(P^t_f;\ U1(out\alpha\, P))\\
\land (Q^t_f;\ U2(out\alpha\, Q))
\end{array}
\right)_{+\{v,tr\}} ;\ M_{\parallel_{cs}}
\right)
\end{array}
\right)
\end{array}
\right\}
$$

[Lemma J.4]

$$
= \left\{ \begin{array}{l} tr' - tr \mid \\ \quad okay \\ \quad \wedge \left(\begin{array}{l} \mathbf{R1} \left(\begin{array}{r} \exists\, 1.tr', 2.tr' \bullet (P_f^f ;\ 1.tr' = tr) \\ \wedge\, (Q_f ;\ 2.tr' = tr) \\ \wedge\, 1.tr' \upharpoonright cs = 2.tr' \upharpoonright cs \end{array} \right) \\ \vee\, \mathbf{R1} \left(\begin{array}{r} \exists\, 1.tr', 2.tr' \bullet (P_f ;\ 1.tr' = tr) \\ \wedge\, (Q_f^f ;\ 2.tr' = tr) \\ \wedge\, 1.tr' \upharpoonright cs = 2.tr' \upharpoonright cs \end{array} \right) \\ \vee \left(\left(\begin{array}{l} (P_f^t ;\ U1(out\alpha\, P)) \\ \wedge\, (Q_f^t ;\ U2(out\alpha\, Q)) \end{array} \right)_{+\{v,tr\}} ;\ M_{\parallel_{cs}} \right) \end{array} \right) \end{array} \right\}
$$

$$
\cup \left\{ \begin{array}{l} (tr' - tr) ^\frown \langle \checkmark \rangle \mid \\ \quad okay \wedge \neg\, wait' \\ \quad \wedge \left(\begin{array}{l} \mathbf{R1} \left(\begin{array}{r} \exists\, 1.tr', 2.tr' \bullet (P_f^f ;\ 1.tr' = tr) \\ \wedge\, (Q_f ;\ 2.tr' = tr) \\ \wedge\, 1.tr' \upharpoonright cs = 2.tr' \upharpoonright cs \end{array} \right) \\ \vee\, \mathbf{R1} \left(\begin{array}{r} \exists\, 1.tr', 2.tr' \bullet (P_f ;\ 1.tr' = tr) \\ \wedge\, (Q_f^f ;\ 2.tr' = tr) \\ \wedge\, 1.tr' \upharpoonright cs = 2.tr' \upharpoonright cs \end{array} \right) \\ \vee \left(\left(\begin{array}{l} (P_f^t ;\ U1(out\alpha\, P)) \\ \wedge\, (Q_f^t ;\ U2(out\alpha\, Q)) \end{array} \right)_{+\{v,tr\}} ;\ M_{\parallel_{cs}} \right) \end{array} \right) \end{array} \right\}
$$

[Assumption 1]

$$= \left\{ \begin{array}{l} tr' - tr \mid \\ \quad okay \\ \quad \wedge \left(\begin{array}{l} \mathbf{R1} \left(\begin{array}{l} \exists\, 1.tr', 2.tr' \bullet (false;\ 1.tr' = tr) \\ \qquad \wedge\ (Q_f;\ 2.tr' = tr) \\ \qquad \wedge\ 1.tr' \upharpoonright cs = 2.tr' \upharpoonright cs \end{array} \right) \\ \vee\ \mathbf{R1} \left(\begin{array}{l} \exists\, 1.tr', 2.tr' \bullet (P_f;\ 1.tr' = tr) \\ \qquad \wedge\ (false;\ 2.tr' = tr) \\ \qquad \wedge\ 1.tr' \upharpoonright cs = 2.tr' \upharpoonright cs \end{array} \right) \\ \vee \left(\left(\begin{array}{l} (P^t_f;\ U1(out\alpha\, P)) \\ \wedge\ (Q^t_f;\ U2(out\alpha\, Q)) \end{array} \right)_{+\{v,tr\}} ;\ M_{\|_{cs}} \right) \end{array} \right) \end{array} \right\}$$

$$\cup \left\{ \begin{array}{l} (tr' - tr) ^\frown \langle \checkmark \rangle \mid \\ \quad okay \wedge \neg\, wait' \\ \quad \wedge \left(\begin{array}{l} \mathbf{R1} \left(\begin{array}{l} \exists\, 1.tr', 2.tr' \bullet (false;\ 1.tr' = tr) \\ \qquad \wedge\ (Q_f;\ 2.tr' = tr) \\ \qquad \wedge\ 1.tr' \upharpoonright cs = 2.tr' \upharpoonright cs \end{array} \right) \\ \vee\ \mathbf{R1} \left(\begin{array}{l} \exists\, 1.tr', 2.tr' \bullet (P_f;\ 1.tr' = tr) \\ \qquad \wedge\ (false;\ 2.tr' = tr) \\ \qquad \wedge\ 1.tr' \upharpoonright cs = 2.tr' \upharpoonright cs \end{array} \right) \\ \vee \left(\left(\begin{array}{l} (P^t_f;\ U1(out\alpha\, P)) \\ \wedge\ (Q^t_f;\ U2(out\alpha\, Q)) \end{array} \right)_{+\{v,tr\}} ;\ M_{\|_{cs}} \right) \end{array} \right) \end{array} \right\}$$

[Sequence and PC]

$$= \left\{ \begin{array}{l} tr' - tr \mid \\ \quad okay \\ \quad \wedge \left(\left(\begin{array}{l} (P^t_f;\ U1(out\alpha\, P)) \\ \wedge\ (Q^t_f;\ U2(out\alpha\, Q)) \end{array} \right)_{+\{v,tr\}} ;\ M_{\|_{cs}} \right) \end{array} \right\}$$

$$\cup \left\{ \begin{array}{l} (tr' - tr) ^\frown \langle \checkmark \rangle \mid \\ \quad okay \wedge \neg\, wait' \\ \quad \wedge \left(\left(\begin{array}{l} (P^t_f;\ U1(out\alpha\, P)) \\ \wedge\ (Q^t_f;\ U2(out\alpha\, Q)) \end{array} \right)_{+\{v,tr\}} ;\ M_{\|_{cs}} \right) \end{array} \right\}$$

$[M_{\|_{cs}}]$

$$
= \left\{ \begin{array}{l}
tr' - tr \mid \\
\quad okay \\
\quad \wedge \left(\begin{array}{l}
\left(\begin{array}{l} (P^t_f;\ U1(out\alpha\, P)) \\ \wedge\ (Q^t_f;\ U2(out\alpha\, Q)) \end{array} \right)_{+\{v,tr\}} ; \\
\left(\begin{array}{l}
tr' - tr \in (1.tr - tr \parallel_{cs} 2.tr - tr) \\
\wedge\ 1.tr \upharpoonright cs = 2.tr \upharpoonright cs \\
\wedge \left(\begin{array}{l}
\left(\begin{array}{l} (1.wait \vee 2.wait) \\ \wedge\ ref' \subseteq \left(\begin{array}{l} ((1.ref \cup 2.ref) \cap cs) \\ \cup((1.ref \cap 2.ref) \setminus cs) \end{array} \right) \end{array} \right) \\
\lhd wait' \rhd \\
(\neg\, 1.wait \wedge \neg\, 2.wait \wedge MSt)
\end{array} \right)
\end{array} \right)
\end{array} \right)
\end{array} \right\}
$$

$$
\cup \left\{ \begin{array}{l}
(tr' - tr) \frown \langle \checkmark \rangle \mid \\
\quad okay \wedge \neg\, wait' \\
\quad \wedge \left(\begin{array}{l}
\left(\begin{array}{l} (P^t_f;\ U1(out\alpha\, P)) \\ \wedge\ (Q^t_f;\ U2(out\alpha\, Q)) \end{array} \right)_{+\{v,tr\}} ; \\
\left(\begin{array}{l}
tr' - tr \in (1.tr - tr \parallel_{cs} 2.tr - tr) \\
\wedge\ 1.tr \upharpoonright cs = 2.tr \upharpoonright cs \\
\wedge \left(\begin{array}{l}
\left(\begin{array}{l} (1.wait \vee 2.wait) \\ \wedge\ ref' \subseteq \left(\begin{array}{l} ((1.ref \cup 2.ref) \cap cs) \\ \cup((1.ref \cap 2.ref) \setminus cs) \end{array} \right) \end{array} \right) \\
\lhd wait' \rhd \\
(\neg\, 1.wait \wedge \neg\, 2.wait \wedge MSt)
\end{array} \right)
\end{array} \right)
\end{array} \right)
\end{array} \right\}
$$

[Sequence, PC and ST]

$$= \left\{ \begin{array}{l} tr' - tr \mid \\ \qquad okay \land wait' \\ \qquad \land \left(\begin{array}{l} \left(\begin{array}{l} (P^t_f;\ U1(out\alpha\, P)) \\ \land\ (Q^t_f;\ U2(out\alpha\, Q)) \end{array} \right)_{+\{v,tr\}} ; \\ \left(\begin{array}{l} tr' - tr \in (1.tr - tr \parallel_{cs} 2.tr - tr) \\ \land\ 1.tr \upharpoonright cs = 2.tr \upharpoonright cs \\ \land\ (1.wait \lor 2.wait) \\ \land\ ref' \subseteq \left(\begin{array}{l} ((1.ref \cup 2.ref) \cap cs) \\ \cup((1.ref \cap 2.ref) \setminus cs) \end{array} \right) \end{array} \right) \end{array} \right) \end{array} \right\}$$

$$\cup \left\{ \begin{array}{l} tr' - tr \mid \\ \qquad okay \land \neg\ wait' \\ \qquad \land \left(\begin{array}{l} \left(\begin{array}{l} (P^t_f;\ U1(out\alpha\, P)) \\ \land\ (Q^t_f;\ U2(out\alpha\, Q)) \end{array} \right)_{+\{v,tr\}} ; \\ \left(\begin{array}{l} tr' - tr \in (1.tr - tr \parallel_{cs} 2.tr - tr) \\ \land\ 1.tr \upharpoonright cs = 2.tr \upharpoonright cs \\ \land\ (\neg\ 1.wait \land \neg\ 2.wait \land MSt) \end{array} \right) \end{array} \right) \end{array} \right\}$$

$$\cup \left\{ \begin{array}{l} (tr' - tr) ^\frown \langle \checkmark \rangle \mid \\ \qquad okay \land \neg\ wait' \\ \qquad \land \left(\begin{array}{l} \left(\begin{array}{l} (P^t_f;\ U1(out\alpha\, P)) \\ \land\ (Q^t_f;\ U2(out\alpha\, Q)) \end{array} \right)_{+\{v,tr\}} ; \\ \left(\begin{array}{l} tr' - tr \in (1.tr - tr \parallel_{cs} 2.tr - tr) \\ \land\ 1.tr \upharpoonright cs = 2.tr \upharpoonright cs \\ \land\ (\neg\ 1.wait \land \neg\ 2.wait \land MSt) \end{array} \right) \end{array} \right) \end{array} \right\}$$

[Lemma J.31]

$$= \left\{ \begin{array}{l} tr' - tr \mid \\ \quad okay \wedge wait' \\ \quad \wedge \left(\begin{array}{l} \left(\begin{array}{l} (P^t_f;\ U1(out\alpha\, P)) \\ \wedge\ (Q^t_f;\ U2(out\alpha\, Q)) \end{array} \right)_{+\{v,tr\}}; \\ \left(\begin{array}{l} tr' - tr \in (1.tr - tr \parallel_{cs} 2.tr - tr) \\ \wedge\ 1.tr \upharpoonright cs = 2.tr \upharpoonright cs \\ \wedge\ (1.wait \vee 2.wait) \\ \wedge\ ref' \subseteq \left(\begin{array}{l} ((1.ref \cup 2.ref) \cap cs) \\ \cup((1.ref \cap 2.ref) \setminus cs) \end{array} \right) \end{array} \right) \end{array} \right) \end{array} \right\}$$
$$\cup \bigcup \left\{ \begin{array}{l} s \underset{cs^{\checkmark}}{\parallel} t \mid s \in \{tr' - tr \mid okay \wedge \neg\, wait' \wedge (P)^t_f\} \\ \qquad \wedge\ t \in \{tr' - tr \mid okay \wedge \neg\, wait' \wedge (Q)^t_f\} \end{array} \right\}$$
$$\cup \left\{ \begin{array}{l} (tr' - tr) \frown \langle \checkmark \rangle \mid \\ \quad okay \wedge \neg\, wait' \\ \quad \wedge \left(\begin{array}{l} \left(\begin{array}{l} (P^t_f;\ U1(out\alpha\, P)) \\ \wedge\ (Q^t_f;\ U2(out\alpha\, Q)) \end{array} \right)_{+\{v,tr\}}; \\ \left(\begin{array}{l} tr' - tr \in (1.tr - tr \parallel_{cs} 2.tr - tr) \\ \wedge\ 1.tr \upharpoonright cs = 2.tr \upharpoonright cs \\ \wedge\ (\neg\, 1.wait \wedge \neg\, 2.wait \wedge MSt) \end{array} \right) \end{array} \right) \end{array} \right\}$$

[Lemma J.32]

$$= \left\{ \begin{array}{l} tr' - tr \mid \\ \quad okay \wedge wait' \\ \quad \wedge \left(\begin{array}{l} \left(\begin{array}{l} (P^t_f;\ U1(out\alpha\, P)) \\ \wedge\ (Q^t_f;\ U2(out\alpha\, Q)) \end{array} \right)_{+\{v,tr\}}; \\ \left(\begin{array}{l} tr' - tr \in (1.tr - tr \parallel_{cs} 2.tr - tr) \\ \wedge\ 1.tr \upharpoonright cs = 2.tr \upharpoonright cs \\ \wedge\ (1.wait \vee 2.wait) \\ \wedge\ ref' \subseteq \left(\begin{array}{l} ((1.ref \cup 2.ref) \cap cs) \\ \cup((1.ref \cap 2.ref) \setminus cs) \end{array} \right) \end{array} \right) \end{array} \right) \end{array} \right\}$$
$$\cup \bigcup \left\{ \begin{array}{l} s \underset{cs^{\checkmark}}{\parallel} t \mid s \in \{tr' - tr \mid okay \wedge \neg\, wait' \wedge (P)^t_f\} \\ \qquad \wedge\ t \in \{tr' - tr \mid okay \wedge \neg\, wait' \wedge (Q)^t_f\} \end{array} \right\}$$
$$\cup \bigcup \left\{ \begin{array}{l} s \frown \langle \checkmark \rangle \underset{cs^{\checkmark}}{\parallel} t \frown \langle \checkmark \rangle \mid s \in \{tr' - tr \mid okay \wedge \neg\, wait' \wedge (P)^t_f\} \\ \qquad\qquad \wedge\ t \in \{tr' - tr \mid okay \wedge \neg\, wait' \wedge (Q)^t_f\} \end{array} \right\}$$

[PC, Sequential Composition, SC, and ST]

$$
= \left\{ \begin{array}{l} tr' - tr \mid \\ \qquad okay \land wait' \\ \qquad \land \left(\begin{array}{l} \left(\begin{array}{l} (P^t_f;\ U1(out\alpha\, P)) \\ \land\ (Q^t_f;\ U2(out\alpha\, Q)) \end{array} \right)_{+\{v,tr\}} ; \\ \left(\begin{array}{l} tr' - tr \in (1.tr - tr \parallel_{cs} 2.tr - tr) \\ \land\ 1.tr \upharpoonright cs = 2.tr \upharpoonright cs \\ \land\ (1.wait \land 2.wait) \\ \land\ ref' \subseteq \left(\begin{array}{l} ((1.ref \cup 2.ref) \cap cs) \\ \cup((1.ref \cap 2.ref) \setminus cs) \end{array} \right) \end{array} \right) \end{array} \right) \end{array} \right\}
$$

$$
\cup \left\{ \begin{array}{l} tr' - tr \mid \\ \qquad okay \land wait' \\ \qquad \land \left(\begin{array}{l} \left(\begin{array}{l} (P^t_f;\ U1(out\alpha\, P)) \\ \land\ (Q^t_f;\ U2(out\alpha\, Q)) \end{array} \right)_{+\{v,tr\}} ; \\ \left(\begin{array}{l} tr' - tr \in (1.tr - tr \parallel_{cs} 2.tr - tr) \\ \land\ 1.tr \upharpoonright cs = 2.tr \upharpoonright cs \\ \land\ (1.wait \land \neg\ 2.wait) \\ \land\ ref' \subseteq \left(\begin{array}{l} ((1.ref \cup 2.ref) \cap cs) \\ \cup((1.ref \cap 2.ref) \setminus cs) \end{array} \right) \end{array} \right) \end{array} \right) \end{array} \right\}
$$

$$
\cup \left\{ \begin{array}{l} tr' - tr \mid \\ \qquad okay \land wait' \\ \qquad \land \left(\begin{array}{l} \left(\begin{array}{l} (P^t_f;\ U1(out\alpha\, P)) \\ \land\ (Q^t_f;\ U2(out\alpha\, Q)) \end{array} \right)_{+\{v,tr\}} ; \\ \left(\begin{array}{l} tr' - tr \in (1.tr - tr \parallel_{cs} 2.tr - tr) \\ \land\ 1.tr \upharpoonright cs = 2.tr \upharpoonright cs \\ \land\ (\neg\ 1.wait \lor 2.wait) \\ \land\ ref' \subseteq \left(\begin{array}{l} ((1.ref \cup 2.ref) \cap cs) \\ \cup((1.ref \cap 2.ref) \setminus cs) \end{array} \right) \end{array} \right) \end{array} \right) \end{array} \right\}
$$

$$
\cup \bigcup \left\{ \begin{array}{l} s \underset{cs^{\checkmark}}{\parallel} t \mid s \in \{tr' - tr \mid okay \land \neg\ wait' \land (P)^t_f\} \\ \qquad \land\ t \in \{tr' - tr \mid okay \land \neg\ wait' \land (Q)^t_f\} \end{array} \right\}
$$

$$
\cup \bigcup \left\{ \begin{array}{l} s \frown \langle \checkmark \rangle \underset{cs^{\checkmark}}{\parallel} t \frown \langle \checkmark \rangle \mid s \in \{tr' - tr \mid okay \land \neg\ wait' \land (P)^t_f\} \\ \qquad\qquad \land\ t \in \{tr' - tr \mid okay \land \neg\ wait' \land (Q)^t_f\} \end{array} \right\}
$$

[Lemmas J.33, J.34, and J.35]

$$
= \bigcup \left\{ \begin{array}{l} s \underset{cs^{\checkmark}}{\parallel} t \mid s \in \{tr' - tr \mid okay \land wait' \land (P)^t_f\} \\ \qquad \land\ t \in \{tr' - tr \mid okay \land wait' \land (Q)^t_f\} \end{array} \right\}
$$

$$
\cup \bigcup \left\{ \begin{array}{l} s \underset{cs^{\checkmark}}{\parallel} t \mid s \in \{tr' - tr \mid okay \land wait' \land (P)^t_f\} \\ \qquad \land\ t \in \{tr' - tr \mid okay \land \neg\ wait' \land (Q)^t_f\} \end{array} \right\}
$$

$$
\cup \bigcup \left\{ \begin{array}{l} s \underset{cs^{\checkmark}}{\parallel} t \mid s \in \{tr' - tr \mid okay \land \neg\ wait' \land (P)^t_f\} \\ \qquad \land\ t \in \{tr' - tr \mid okay \land wait' \land (Q)^t_f\} \end{array} \right\}
$$

$$\cup \bigcup \left\{ \begin{array}{l} s \underset{cs^{\checkmark}}{\parallel} t \mid s \in \{tr' - tr \mid okay \land \neg\, wait' \land (P)^t_f\} \\ \qquad \land\; t \in \{tr' - tr \mid okay \land \neg\, wait' \land (Q)^t_f\} \end{array} \right\}$$

$$\cup \bigcup \left\{ \begin{array}{l} s \frown \langle \checkmark \rangle \underset{cs^{\checkmark}}{\parallel} t \frown \langle \checkmark \rangle \mid s \in \{tr' - tr \mid okay \land \neg\, wait' \land (P)^t_f\} \\ \qquad\qquad\qquad \land\; t \in \{tr' - tr \mid okay \land \neg\, wait' \land (Q)^t_f\} \end{array} \right\}$$

[ST]

$$= \bigcup \left\{ \begin{array}{l} s \underset{cs^{\checkmark}}{\parallel} t \mid s \in \{tr' - tr \mid okay \land wait' \land (P)^t_f\} \\ \qquad \land\; t \in \{tr' - tr \mid okay \land wait' \land (Q)^t_f\} \end{array} \right\}$$

$$\cup \bigcup \left\{ \begin{array}{l} s \underset{cs^{\checkmark}}{\parallel} t \mid s \in \{tr' - tr \mid okay \land wait' \land (P)^t_f\} \\ \qquad \land\; t \in \{tr' - tr \mid okay \land \neg\, wait' \land (Q)^t_f\} \end{array} \right\}$$

$$\cup \bigcup \left\{ \begin{array}{l} s \underset{cs^{\checkmark}}{\parallel} t \mid s \in \{tr' - tr \mid okay \land \neg\, wait' \land (P)^t_f\} \\ \qquad \land\; t \in \{tr' - tr \mid okay \land wait' \land (Q)^t_f\} \end{array} \right\}$$

$$\cup \bigcup \left\{ \begin{array}{l} s \underset{cs^{\checkmark}}{\parallel} t \mid s \in \{tr' - tr \mid okay \land \neg\, wait' \land (P)^t_f\} \\ \qquad \land\; t \in \{tr' - tr \mid okay \land \neg\, wait' \land (Q)^t_f\} \end{array} \right\}$$

$$\cup \bigcup \{\{\}\}$$
$$\cup \bigcup \{\{\}\}$$
$$\cup \bigcup \{\{\}\}$$
$$\cup \bigcup \{\{\}\}$$

$$\cup \bigcup \left\{ \begin{array}{l} s \frown \langle \checkmark \rangle \underset{cs^{\checkmark}}{\parallel} t \frown \langle \checkmark \rangle \mid s \in \{tr' - tr \mid okay \land \neg\, wait' \land (P)^t_f\} \\ \qquad\qquad\qquad \land\; t \in \{tr' - tr \mid okay \land \neg\, wait' \land (Q)^t_f\} \end{array} \right\}$$

[Lemma J.48 (tr, tr' : seq Σ and $\checkmark \notin \Sigma$), SC and ST]

$$= \bigcup \left\{ \begin{array}{l} s \underset{cs^{\checkmark}}{\parallel} t \mid s \in \{tr' - tr \mid okay \land wait' \land (P)^t_f\} \\ \qquad \land\; t \in \{tr' - tr \mid okay \land wait' \land (Q)^t_f\} \end{array} \right\}$$

$$\cup \bigcup \left\{ \begin{array}{l} s \underset{cs^{\checkmark}}{\parallel} t \mid s \in \{tr' - tr \mid okay \land wait' \land (P)^t_f\} \\ \qquad \land\; t \in \{tr' - tr \mid okay \land \neg\, wait' \land (Q)^t_f\} \end{array} \right\}$$

$$\cup \bigcup \left\{ \begin{array}{l} s \underset{cs^{\checkmark}}{\parallel} t \mid s \in \{tr' - tr \mid okay \land \neg\, wait' \land (P)^t_f\} \\ \qquad \land\; t \in \{tr' - tr \mid okay \land wait' \land (Q)^t_f\} \end{array} \right\}$$

$$\cup \bigcup \left\{ \begin{array}{l} s \underset{cs^{\checkmark}}{\parallel} t \mid s \in \{tr' - tr \mid okay \land \neg\, wait' \land (P)^t_f\} \\ \qquad \land\; t \in \{tr' - tr \mid okay \land \neg\, wait' \land (Q)^t_f\} \end{array} \right\}$$

$$\cup \bigcup \left\{ \begin{array}{l} s \underset{cs^{\checkmark}}{\parallel} t \frown \langle \checkmark \rangle \mid s \in \{tr' - tr \mid okay \land wait' \land (P)^t_f\} \\ \qquad\qquad \land\; t \in \{tr' - tr \mid okay \land \neg\, wait' \land (Q)^t_f\} \end{array} \right\}$$

$$\cup \bigcup \left\{ \begin{array}{l} s \underset{cs^{\checkmark}}{\parallel} t \frown \langle \checkmark \rangle \mid s \in \{tr' - tr \mid okay \land \neg\, wait' \land (P)^t_f\} \\ \qquad\qquad \land\; t \in \{tr' - tr \mid okay \land \neg\, wait' \land (Q)^t_f\} \end{array} \right\}$$

$$\cup \bigcup \left\{ \begin{array}{l} s \frown \langle \checkmark \rangle \underset{cs^{\checkmark}}{\parallel} t \mid s \in \{tr' - tr \mid okay \land \neg\, wait' \land (P)^t_f\} \\ \qquad\qquad \land\; t \in \{tr' - tr \mid okay \land wait' \land (Q)^t_f\} \end{array} \right\}$$

$$\cup \bigcup \left\{ \begin{array}{l} s \frown \langle \checkmark \rangle \underset{cs^{\checkmark}}{\parallel} t \mid s \in \{ tr' - tr \mid okay \land \neg\, wait' \land (P)^t_f \} \\ \qquad \land\, t \in \{ tr' - tr \mid okay \land \neg\, wait' \land (Q)^t_f \} \end{array} \right\}$$
$$\cup \bigcup \left\{ \begin{array}{l} s \frown \langle \checkmark \rangle \underset{cs^{\checkmark}}{\parallel} t \frown \langle \checkmark \rangle \mid s \in \{ tr' - tr \mid okay \land \neg\, wait' \land (P)^t_f \} \\ \qquad \land\, t \in \{ tr' - tr \mid okay \land \neg\, wait' \land (Q)^t_f \} \end{array} \right\}$$

[SC]

$$= \bigcup \left\{ \begin{array}{l} s \underset{cs^{\checkmark}}{\parallel} t \mid s \in \{ tr' - tr \mid okay \land wait' \land (P)^t_f \} \\ \qquad \land\, t \in \{ tr' - tr \mid okay \land wait' \land (Q)^t_f \} \end{array} \right\}$$
$$\cup \bigcup \left\{ \begin{array}{l} s \underset{cs^{\checkmark}}{\parallel} t \mid s \in \{ tr' - tr \mid okay \land wait' \land (P)^t_f \} \\ \qquad \land\, t \in \{ tr' - tr \mid okay \land \neg\, wait' \land (Q)^t_f \} \end{array} \right\}$$
$$\cup \bigcup \left\{ \begin{array}{l} s \underset{cs^{\checkmark}}{\parallel} t \mid s \in \{ tr' - tr \mid okay \land \neg\, wait' \land (P)^t_f \} \\ \qquad \land\, t \in \{ tr' - tr \mid okay \land wait' \land (Q)^t_f \} \end{array} \right\}$$
$$\cup \bigcup \left\{ \begin{array}{l} s \underset{cs^{\checkmark}}{\parallel} t \mid s \in \{ tr' - tr \mid okay \land \neg\, wait' \land (P)^t_f \} \\ \qquad \land\, t \in \{ tr' - tr \mid okay \land \neg\, wait' \land (Q)^t_f \} \end{array} \right\}$$
$$\cup \bigcup \left\{ \begin{array}{l} s \underset{cs^{\checkmark}}{\parallel} t \mid s \in \{ tr' - tr \mid okay \land wait' \land (P)^t_f \} \\ \qquad \land\, t \in \{ (tr' - tr) \frown \langle \checkmark \rangle \mid okay \land \neg\, wait' \land (Q)^t_f \} \end{array} \right\}$$
$$\cup \bigcup \left\{ \begin{array}{l} s \underset{cs^{\checkmark}}{\parallel} t \mid s \in \{ tr' - tr \mid okay \land \neg\, wait' \land (P)^t_f \} \\ \qquad \land\, t \in \{ (tr' - tr) \frown \langle \checkmark \rangle \mid okay \land \neg\, wait' \land (Q)^t_f \} \end{array} \right\}$$
$$\cup \bigcup \left\{ \begin{array}{l} s \underset{cs^{\checkmark}}{\parallel} t \mid s \in \{ (tr' - tr) \frown \langle \checkmark \rangle \mid okay \land \neg\, wait' \land (P)^t_f \} \\ \qquad \land\, t \in \{ tr' - tr \mid okay \land wait' \land (Q)^t_f \} \end{array} \right\}$$
$$\cup \bigcup \left\{ \begin{array}{l} s \underset{cs^{\checkmark}}{\parallel} t \mid s \in \{ (tr' - tr) \frown \langle \checkmark \rangle \mid okay \land \neg\, wait' \land (P)^t_f \} \\ \qquad \land\, t \in \{ tr' - tr \mid okay \land \neg\, wait' \land (Q)^t_f \} \end{array} \right\}$$
$$\cup \bigcup \left\{ \begin{array}{l} s \underset{cs^{\checkmark}}{\parallel} t \mid s \in \{ (tr' - tr) \frown \langle \checkmark \rangle \mid okay \land \neg\, wait' \land (P)^t_f \} \\ \qquad \land\, t \in \{ (tr' - tr) \frown \langle \checkmark \rangle \mid okay \land \neg\, wait' \land (Q)^t_f \} \end{array} \right\}$$

[PC, SC, and ST]

$$= \bigcup \left\{ \begin{array}{l} s \underset{cs^{\checkmark}}{\parallel} t \mid s \in \{ tr' - tr \mid okay \land (wait' \lor \neg\, wait') \land (P)^t_f \} \\ \qquad \land\, t \in \{ tr' - tr \mid okay \land (wait' \lor \neg\, wait') \land (Q)^t_f \} \end{array} \right\}$$
$$\cup \bigcup \left\{ \begin{array}{l} s \underset{cs^{\checkmark}}{\parallel} t \mid s \in \{ tr' - tr \mid okay \land (wait' \lor \neg\, wait') \land (P)^t_f \} \\ \qquad \land\, t \in \{ (tr' - tr) \frown \langle \checkmark \rangle \mid okay \land \neg\, wait' \land (Q)^t_f \} \end{array} \right\}$$
$$\cup \bigcup \left\{ \begin{array}{l} s \underset{cs^{\checkmark}}{\parallel} t \mid s \in \{ (tr' - tr) \frown \langle \checkmark \rangle \mid okay \land \neg\, wait' \land (P)^t_f \} \\ \qquad \land\, t \in \{ tr' - tr \mid okay \land (wait' \lor \neg\, wait') \land (Q)^t_f \} \end{array} \right\}$$
$$\cup \bigcup \left\{ \begin{array}{l} s \underset{cs^{\checkmark}}{\parallel} t \mid s \in \{ (tr' - tr) \frown \langle \checkmark \rangle \mid okay \land \neg\, wait' \land (P)^t_f \} \\ \qquad \land\, t \in \{ (tr' - tr) \frown \langle \checkmark \rangle \mid okay \land \neg\, wait' \land (Q)^t_f \} \end{array} \right\}$$

[PC]

$$= \bigcup \left\{ \begin{array}{l} s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \mid s \in \{tr' - tr \mid okay \wedge (P)^t_f\} \\ \quad \wedge\ t \in \{tr' - tr \mid okay \wedge (Q)^t_f\} \end{array} \right\}$$

$$\cup \bigcup \left\{ \begin{array}{l} s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \mid s \in \{tr' - tr \mid okay \wedge (P)^t_f\} \\ \quad \wedge\ t \in \{(tr' - tr) \frown \langle \checkmark \rangle \mid okay \wedge \neg\, wait' \wedge (Q)^t_f\} \end{array} \right\}$$

$$\cup \bigcup \left\{ \begin{array}{l} s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \mid s \in \{(tr' - tr) \frown \langle \checkmark \rangle \mid okay \wedge \neg\, wait' \wedge (P)^t_f\} \\ \quad \wedge\ t \in \{tr' - tr \mid okay \wedge (Q)^t_f\} \end{array} \right\}$$

$$\cup \bigcup \left\{ \begin{array}{l} s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \mid s \in \{(tr' - tr) \frown \langle \checkmark \rangle \mid okay \wedge \neg\, wait' \wedge (P)^t_f\} \\ \quad \wedge\ t \in \{(tr' - tr) \frown \langle \checkmark \rangle \mid okay \wedge \neg\, wait' \wedge (Q)^t_f\} \end{array} \right\}$$

[ST, SC and PC]

$$= \bigcup \left\{ \begin{array}{l} s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \mid s \in \left(\begin{array}{l} \{tr' - tr \mid okay \wedge (P)^t_f\} \\ \cup \{(tr' - tr) \frown \langle \checkmark \rangle \mid okay \wedge \neg\, wait' \wedge (P)^t_f\} \end{array} \right) \\ \qquad \wedge\ t \in \left(\begin{array}{l} \{tr' - tr \mid okay \wedge (Q)^t_f\} \\ \cup \{(tr' - tr) \frown \langle \checkmark \rangle \mid okay \wedge \neg\, wait' \wedge (Q)^t_f\} \end{array} \right) \end{array} \right\} \quad [PC]$$

$$= \bigcup \left\{ \begin{array}{l} s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \mid s \in \left(\begin{array}{l} \{tr' - tr \mid okay \wedge \neg\, wait \wedge okay' \wedge P\} \\ \cup \{(tr' - tr) \frown \langle \checkmark \rangle \mid okay \wedge \neg\, wait \wedge okay' \wedge \neg\, wait' \wedge P\} \end{array} \right) \\ \qquad \wedge\ t \in \left(\begin{array}{l} \{tr' - tr \mid okay \wedge \neg\, wait \wedge okay' \wedge Q\} \\ \cup \{(tr' - tr) \frown \langle \checkmark \rangle \mid okay \wedge \neg\, wait \wedge okay' \wedge \neg\, wait' \wedge Q\} \end{array} \right) \end{array} \right\}$$

$[A^n]$

$$= \bigcup \left\{ \begin{array}{l} s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \mid s \in \left(\begin{array}{l} \{tr' - tr \mid (P)^n\} \\ \cup \{(tr' - tr) \frown \langle \checkmark \rangle \mid \neg\, wait' \wedge (P)^n\} \end{array} \right) \\ \qquad \wedge\ t \in \left(\begin{array}{l} \{tr' - tr \mid (Q)^n\} \\ \cup \{(tr' - tr) \frown \langle \checkmark \rangle \mid \neg\, wait' \wedge (Q)^n\} \end{array} \right) \end{array} \right\} \quad [A^t]$$

$$= \bigcup \left\{ \begin{array}{l} s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \mid s \in \left(\begin{array}{l} \{tr' - tr \mid (P)^n\} \\ \cup \{(tr' - tr) \frown \langle \checkmark \rangle \mid (P)^t\} \end{array} \right) \\ \qquad \wedge\ t \in \left(\begin{array}{l} \{tr' - tr \mid (Q)^n\} \\ \cup \{(tr' - tr) \frown \langle \checkmark \rangle \mid (Q)^t\} \end{array} \right) \end{array} \right\} \quad [traces^{\mathcal{UTP}}]$$

$$= \bigcup \{ s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \mid s \in traces^{\mathcal{UTP}}(P) \wedge t \in traces^{\mathcal{UTP}}(Q)\} \quad [IH]$$

$$= \bigcup \{ s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \mid s \in traces(\Upsilon(P)) \wedge t \in traces(\Upsilon(Q))\} \quad [traces]$$

$$= traces(\Upsilon(P) \underset{\mathsf{cs}}{\parallel} \Upsilon(Q)) \quad [\text{Notation } (\Upsilon_{\mathbb{P}^{cs}}(cs) = \mathsf{cs})]$$

$$= traces(\Upsilon(P) \underset{\Upsilon_{\mathbb{P}}(cs)}{\parallel} \Upsilon(Q)) \quad [\Upsilon]$$

$$= traces(\Upsilon(P \llbracket ns_1 \mid cs \mid ns_2 \rrbracket Q))$$

Theorem J.18

$$traces^{\mathcal{UTP}}(\llbracket cs \rrbracket\ x : S \bullet \llbracket ns \rrbracket\ A) = traces(\Upsilon(\llbracket cs \rrbracket\ x : S \bullet \llbracket ns \rrbracket\ A))$$

provided

1. $\forall i : S \bullet A[v_i/x]$ *is* $\boldsymbol{R}$
2. $\forall i : S \bullet A[v_i/x]$ *is divergence-free*
3. $S \neq \{\}$

Inductive Hypothesis (A):

$$\forall i : S \bullet traces^{\mathcal{UTP}}(A[v_i/x]) = traces(\Upsilon(A)[v_i/x])$$

Proof. By induction on S

Base Case. $S = \{v\}$
Proof.

$$\begin{aligned}
& traces^{\mathcal{UTP}}(\|[cs]\|\, x : S \bullet \|[ns]\|\, A) && \text{[Assumption]}\\
&= traces^{\mathcal{UTP}}(\|[cs]\|\, x : \{v\} \bullet \|[ns]\|\, A) && \text{[Indexed parallel]}\\
&= traces^{\mathcal{UTP}}(A[v/x]) && \text{[IH]}\\
&= traces(\Upsilon(A[v/x])) && \text{[Indexed parallel]}\\
&= traces(\Upsilon(\|[cs]\|\, x : \{v\} \bullet \|[ns]\|\, A)) && \text{[Assumption]}\\
&= traces(\Upsilon(\|[cs]\|\, x : S \bullet \|[ns]\|\, A))
\end{aligned}$$

Inductive Hypothesis (S):

$$traces^{\mathcal{UTP}}(\|[cs]\|\, x : S \bullet \|[ns]\|\, A) = traces(\Upsilon(\|[cs]\|\, x : S \bullet \|[ns]\|\, A))$$

Inductive Step

$$traces^{\mathcal{UTP}}(\|[cs]\|\, x : S \cup \{v_i\} \bullet \|[ns]\|\, A) = traces(\Upsilon(\|[cs]\|\, x : S \cup \{v_i\} \bullet \|[ns]\|\, A))$$

Proof.

$$\begin{aligned}
& traces^{\mathcal{UTP}}(\|[cs]\|\, x : S \cup \{v_i\} \bullet \|[ns]\|\, A) && \text{[Indexed parallel]}\\
&= traces^{\mathcal{UTP}}(A[v_i/x] \;[\![\, ns[v_i/x] \mid cs \mid \textstyle\bigcup_{v:S\setminus\{v_i\}} ns[v/x] \,]\!]\; (\|[cs]\|\, x : S \setminus \{v_i\} \bullet \|[ns]\|\, A))\\
& && \text{[Theorem J.17 (Provisos, IH-}A\text{ and IH-}S)]\\
&= traces(\Upsilon(A[v_i/x] \;[\![\, ns[v_i/x] \mid cs \mid \textstyle\bigcup_{v:S\setminus\{v_i\}} ns[v/x] \,]\!]\; (\|[cs]\|\, x : S \setminus \{v_i\} \bullet \|[ns]\|\, A)))\\
& && \text{[Indexed parallel]}\\
&= traces(\Upsilon(\|[cs]\|\, x : S \cup \{v_i\} \bullet \|[ns]\|\, A))
\end{aligned}$$

Theorem J.19

$$\begin{array}{l} \mathit{traces}^{\mathcal{UTP}}(P \parallel[ns_1 \mid ns_2]\parallel Q) \\ = \\ \mathit{traces}(\Upsilon(P \parallel[ns_1 \mid ns_2]\parallel Q)) \end{array}$$

provided

1. *P and Q are divergence-free*

Proof.

$$\begin{array}{ll} \mathit{traces}^{\mathcal{UTP}}(P \parallel[ns_1 \mid ns_2]\parallel Q) & [\text{Law } 29] \\ = \mathit{traces}^{\mathcal{UTP}}(P \llbracket\, ns_1 \mid \emptyset \mid ns_2 \,\rrbracket\, Q) & [\text{Theorem J.17 (proviso)}] \\ = \mathit{traces}(\Upsilon(P \llbracket\, ns_1 \mid \emptyset \mid ns_2 \,\rrbracket\, Q)) & [\text{Law } 29] \\ = \mathit{traces}(\Upsilon(P \parallel[ns_1 \mid ns_2]\parallel Q)) & \end{array}$$

Theorem J.20

$$\mathit{traces}^{\mathcal{UTP}}(\interleave\, x : S \bullet \llbracket ns \rrbracket\, A) = \mathit{traces}(\Upsilon(\interleave\, x : S \bullet \llbracket ns \rrbracket\, A))$$

provided

1. $\forall\, i : S \bullet A[v_i/x]$ *is* $\boldsymbol{R}$
2. $\forall\, i : S \bullet A[v_i/x]$ *is divergence-free*
3. $S \neq \{\}$

Proof.

$$\begin{array}{ll} \mathit{traces}^{\mathcal{UTP}}(\interleave\, x : S \bullet \llbracket ns \rrbracket\, A) & [\text{Law } 29] \\ = \mathit{traces}^{\mathcal{UTP}}(\llbracket \emptyset \rrbracket\, x : S \bullet \llbracket ns \rrbracket\, A)) & [\text{Theorem J.17 (proviso)}] \\ = \mathit{traces}(\Upsilon(\llbracket \emptyset \rrbracket\, x : S \bullet \llbracket ns \rrbracket\, A)) & [\text{Law } 29] \\ = \mathit{traces}(\Upsilon(\interleave\, x : S \bullet \llbracket ns \rrbracket\, A)) & \end{array}$$

Theorem J.21 $\mathit{failures}^{\mathcal{UTP}}(\mathit{Skip}) = \mathit{failures}(\Upsilon(\mathit{Skip}))$

Proof.

$$
\begin{aligned}
& \mathit{failures}^{\mathcal{UTP}}(\mathit{Skip}) && [\mathit{failures}^{\mathcal{UTP}}]\\
&= \{(\mathit{tr}' - \mathit{tr}, \mathit{ref}') \mid (\mathit{Skip})^n\} && [A^t]\\
&\quad \cup \{(\mathit{tr}' - \mathit{tr}, \mathit{ref}' \cup \{\checkmark\}) \mid (\mathit{Skip})^n \land \mathit{wait}'\}\\
&\quad \cup \{((\mathit{tr}' - \mathit{tr}) \frown \langle \checkmark \rangle, \mathit{ref}') \mid (\mathit{Skip})^t\}\\
&\quad \cup \{((\mathit{tr}' - \mathit{tr}) \frown \langle \checkmark \rangle, \mathit{ref}' \cup \{\checkmark\}) \mid (\mathit{Skip})^t\}\\
&= \{(\mathit{tr}' - \mathit{tr}, \mathit{ref}') \mid (\mathit{Skip})^n\} && [A^n]\\
&\quad \cup \{(\mathit{tr}' - \mathit{tr}, \mathit{ref}' \cup \{\checkmark\}) \mid (\mathit{Skip})^n \land \mathit{wait}'\}\\
&\quad \cup \{((\mathit{tr}' - \mathit{tr}) \frown \langle \checkmark \rangle, \mathit{ref}') \mid \neg\, \mathit{wait}' \land (\mathit{Skip})^n\}\\
&\quad \cup \{((\mathit{tr}' - \mathit{tr}) \frown \langle \checkmark \rangle, \mathit{ref}' \cup \{\checkmark\}) \mid \neg\, \mathit{wait}' \land (\mathit{Skip})^n\}\\
&= \{(\mathit{tr}' - \mathit{tr}, \mathit{ref}') \mid \mathit{okay} \land \neg\, \mathit{wait} \land \mathit{okay}' \land \mathit{Skip}\} && [\mathrm{PC}]\\
&\quad \cup \{(\mathit{tr}' - \mathit{tr}, \mathit{ref}' \cup \{\checkmark\}) \mid \mathit{okay} \land \neg\, \mathit{wait} \land \mathit{okay}' \land \mathit{wait}' \land \mathit{Skip}\}\\
&\quad \cup \{((\mathit{tr}' - \mathit{tr}) \frown \langle \checkmark \rangle, \mathit{ref}') \mid \mathit{okay} \land \neg\, \mathit{wait} \land \mathit{okay}' \land \neg\, \mathit{wait}' \land \mathit{Skip}\}\\
&\quad \cup \{((\mathit{tr}' - \mathit{tr}) \frown \langle \checkmark \rangle, \mathit{ref}' \cup \{\checkmark\}) \mid \mathit{okay} \land \neg\, \mathit{wait} \land \mathit{okay}' \land \neg\, \mathit{wait}' \land \mathit{Skip}\}\\
&= \{(\mathit{tr}' - \mathit{tr}, \mathit{ref}') \mid \mathit{okay} \land (\mathit{Skip})^t_f\} && [\text{Lemma J.12}]\\
&\quad \cup \{(\mathit{tr}' - \mathit{tr}, \mathit{ref}' \cup \{\checkmark\}) \mid \mathit{okay} \land \mathit{wait}' \land (\mathit{Skip})^t_f\}\\
&\quad \cup \{((\mathit{tr}' - \mathit{tr}) \frown \langle \checkmark \rangle, \mathit{ref}') \mid \mathit{okay} \land \neg\, \mathit{wait}' \land (\mathit{Skip})^t_f\}\\
&\quad \cup \{((\mathit{tr}' - \mathit{tr}) \frown \langle \checkmark \rangle, \mathit{ref}' \cup \{\checkmark\}) \mid \mathit{okay} \land \neg\, \mathit{wait}' \land (\mathit{Skip})^t_f\}\\
&= \{(\mathit{tr}' - \mathit{tr}, \mathit{ref}') \mid \mathit{okay} \land \mathbf{CSP1}(\mathit{tr}' = \mathit{tr} \land \neg\, \mathit{wait}' \land v' = v)\} && [\text{Lemma J.4}]\\
&\quad \cup \{(\mathit{tr}' - \mathit{tr}, \mathit{ref}' \cup \{\checkmark\}) \mid \mathit{okay} \land \mathit{wait}' \land \mathbf{CSP1}(\mathit{tr}' = \mathit{tr} \land \neg\, \mathit{wait}' \land v' = v)\}\\
&\quad \cup \{((\mathit{tr}' - \mathit{tr}) \frown \langle \checkmark \rangle, \mathit{ref}') \mid \mathit{okay} \land \neg\, \mathit{wait}' \land \mathbf{CSP1}(\mathit{tr}' = \mathit{tr} \land \neg\, \mathit{wait}' \land v' = v)\}\\
&\quad \cup \{((\mathit{tr}' - \mathit{tr}) \frown \langle \checkmark \rangle, \mathit{ref}' \cup \{\checkmark\}) \mid \mathit{okay} \land \neg\, \mathit{wait}' \land \mathbf{CSP1}(\mathit{tr}' = \mathit{tr} \land \neg\, \mathit{wait}' \land v' = v)\}\\
&= \{(\mathit{tr}' - \mathit{tr}, \mathit{ref}') \mid \mathit{okay} \land \mathit{tr}' = \mathit{tr} \land \neg\, \mathit{wait}' \land v' = v\} && [\text{PC and SC}]\\
&\quad \cup \{(\mathit{tr}' - \mathit{tr}, \mathit{ref}' \cup \{\checkmark\}) \mid \mathit{okay} \land \mathit{wait}' \land \mathit{tr}' = \mathit{tr} \land \neg\, \mathit{wait}' \land v' = v\}\\
&\quad \cup \{((\mathit{tr}' - \mathit{tr}) \frown \langle \checkmark \rangle, \mathit{ref}') \mid \mathit{okay} \land \neg\, \mathit{wait}' \land \mathit{tr}' = \mathit{tr} \land \neg\, \mathit{wait}' \land v' = v\}\\
&\quad \cup \{((\mathit{tr}' - \mathit{tr}) \frown \langle \checkmark \rangle, \mathit{ref}' \cup \{\checkmark\}) \mid \mathit{okay} \land \neg\, \mathit{wait}' \land \mathit{tr}' = \mathit{tr} \land \neg\, \mathit{wait}' \land v' = v\}\\
&= \{(\mathit{tr}' - \mathit{tr}, \mathit{ref}') \mid \mathit{okay} \land \neg\, \mathit{wait}' \land \mathit{tr}' = \mathit{tr} \land v' = v\} && [\text{SS. and } -]\\
&\quad \cup \{((\mathit{tr}' - \mathit{tr}) \frown \langle \checkmark \rangle, \mathit{ref}') \mid \mathit{okay} \land \neg\, \mathit{wait}' \land \mathit{tr}' = \mathit{tr} \land v' = v\}\\
&\quad \cup \{((\mathit{tr}' - \mathit{tr}) \frown \langle \checkmark \rangle, \mathit{ref}' \cup \{\checkmark\}) \mid \mathit{okay} \land \neg\, \mathit{wait}' \land \mathit{tr}' = \mathit{tr} \land v' = v\}\\
&= \{(\langle\rangle, \mathit{ref}') \mid \mathit{okay} \land \neg\, \mathit{wait}' \land \mathit{tr}' = \mathit{tr} \land v' = v\} && [\text{Cases}]\\
&\quad \cup \{(\langle \checkmark \rangle, \mathit{ref}') \mid \mathit{okay} \land \neg\, \mathit{wait}' \land \mathit{tr}' = \mathit{tr} \land v' = v\}\\
&\quad \cup \{(\langle \checkmark \rangle, \mathit{ref}' \cup \{\checkmark\}) \mid \mathit{okay} \land \neg\, \mathit{wait}' \land \mathit{tr}' = \mathit{tr} \land v' = v\}\\
&= \{(\langle\rangle, \mathit{ref}')\} \cup \{\} \cup \{(\langle \checkmark \rangle, \mathit{ref}')\} \cup \{\} \cup \{(\langle \checkmark \rangle, \mathit{ref}' \cup \{\checkmark\})\} \cup \{\} && [\mathrm{ST}]\\
&= \{(\langle\rangle, \mathit{ref}')\} \cup \{(\langle \checkmark \rangle, \mathit{ref}')\} \cup \{(\langle \checkmark \rangle, \mathit{ref}' \cup \{\checkmark\})\} && [\mathit{ref}' \text{ definition}]\\
&= \{(\langle\rangle, X) \mid X \subseteq \Sigma\} \cup \{(\langle \checkmark \rangle, X) \mid X \subseteq \Sigma\} \cup \{(\langle \checkmark \rangle, X \cup \{\checkmark\}) \mid X \subseteq \Sigma\} && [\mathrm{ST}]\\
&= \{(\langle\rangle, X) \mid X \subseteq \Sigma\} \cup \{(\langle \checkmark \rangle, X) \mid X \subseteq \Sigma \cup \{\checkmark\}\} && [\Sigma^{\checkmark}]
\end{aligned}
$$

$$
\begin{aligned}
&= \{(\langle\rangle, X) \mid X \subseteq \Sigma\} \cup \{(\langle \checkmark \rangle, X) \mid X \subseteq \Sigma^{\checkmark}\} && [\mathit{failures}]\\
&= \mathit{failures}(\texttt{SKIP}) && [\Upsilon]\\
&= \mathit{failures}(\Upsilon(\mathit{Skip}))
\end{aligned}
$$

Theorem J.22 $\mathit{failures}^{\mathcal{UTP}}(\mathit{Stop}) = \mathit{failures}(\Upsilon(\mathit{Stop}))$

Proof.

$$
\begin{aligned}
&\mathit{failures}^{\mathcal{UTP}}(\mathit{Stop}) && [\mathit{failures}^{\mathcal{UTP}}]\\
&= \{(tr' - tr, ref') \mid (\mathit{Stop})^n\} && [A^t]\\
&\quad \cup \{(tr' - tr, ref' \cup \{\checkmark\}) \mid (\mathit{Stop})^n \wedge wait'\}\\
&\quad \cup \{((tr' - tr) \frown \langle \checkmark \rangle, ref') \mid (\mathit{Stop})^t\}\\
&\quad \cup \{((tr' - tr) \frown \langle \checkmark \rangle, ref' \cup \{\checkmark\}) \mid (\mathit{Stop})^t\}\\
&= \{(tr' - tr, ref') \mid (\mathit{Stop})^n\} && [A^n]\\
&\quad \cup \{(tr' - tr, ref' \cup \{\checkmark\}) \mid (\mathit{Stop})^n \wedge wait'\}\\
&\quad \cup \{((tr' - tr) \frown \langle \checkmark \rangle, ref') \mid \neg\, wait' \wedge (\mathit{Stop})^n\}\\
&\quad \cup \{((tr' - tr) \frown \langle \checkmark \rangle, ref' \cup \{\checkmark\}) \mid \neg\, wait' \wedge (\mathit{Stop})^n\}\\
&= \{(tr' - tr, ref') \mid okay \wedge \neg\, wait \wedge okay' \wedge \mathit{Stop}\} && [\text{PC}]\\
&\quad \cup \{(tr' - tr, ref' \cup \{\checkmark\}) \mid okay \wedge \neg\, wait \wedge okay' \wedge wait' \wedge \mathit{Stop}\}\\
&\quad \cup \{((tr' - tr) \frown \langle \checkmark \rangle, ref') \mid okay \wedge \neg\, wait \wedge okay' \wedge \neg\, wait' \wedge \mathit{Stop}\}\\
&\quad \cup \{((tr' - tr) \frown \langle \checkmark \rangle, ref' \cup \{\checkmark\}) \mid okay \wedge \neg\, wait \wedge okay' \wedge \neg\, wait' \wedge \mathit{Stop}\}\\
&= \{(tr' - tr, ref') \mid okay \wedge (\mathit{Stop})^t_f\} && [\text{Lemma J.12}]\\
&\quad \cup \{(tr' - tr, ref' \cup \{\checkmark\}) \mid okay \wedge wait' \wedge (\mathit{Stop})^t_f\}\\
&\quad \cup \{((tr' - tr) \frown \langle \checkmark \rangle, ref') \mid okay \wedge \neg\, wait' \wedge (\mathit{Stop})^t_f\}\\
&\quad \cup \{((tr' - tr) \frown \langle \checkmark \rangle, ref' \cup \{\checkmark\}) \mid okay \wedge \neg\, wait' \wedge (\mathit{Stop})^t_f\}\\
&= \{(tr' - tr, ref') \mid okay \wedge \mathbf{CSP1}(tr' = tr \wedge wait')\} && [\text{Lemma J.4}]\\
&\quad \cup \{(tr' - tr, ref' \cup \{\checkmark\}) \mid okay \wedge wait' \wedge \mathbf{CSP1}(tr' = tr \wedge wait')\}\\
&\quad \cup \{((tr' - tr) \frown \langle \checkmark \rangle, ref') \mid okay \wedge \neg\, wait' \wedge \mathbf{CSP1}(tr' = tr \wedge wait')\}\\
&\quad \cup \{((tr' - tr) \frown \langle \checkmark \rangle, ref' \cup \{\checkmark\}) \mid okay \wedge \neg\, wait' \wedge \mathbf{CSP1}(tr' = tr \wedge wait')\}\\
&= \{(tr' - tr, ref') \mid okay \wedge tr' = tr \wedge wait'\} && [\text{PC and SC}]\\
&\quad \cup \{(tr' - tr, ref' \cup \{\checkmark\}) \mid okay \wedge wait' \wedge tr' = tr \wedge wait'\}\\
&\quad \cup \{((tr' - tr) \frown \langle \checkmark \rangle, ref') \mid okay \wedge \neg\, wait' \wedge tr' = tr \wedge wait'\}\\
&\quad \cup \{((tr' - tr) \frown \langle \checkmark \rangle, ref' \cup \{\checkmark\}) \mid okay \wedge \neg\, wait' \wedge tr' = tr \wedge wait'\}\\
&= \{(tr' - tr, ref') \mid okay \wedge tr' = tr \wedge wait'\} && [\text{SS. and } -]\\
&\quad \cup \{(tr' - tr, ref' \cup \{\checkmark\}) \mid okay \wedge wait' \wedge tr' = tr\}\\
&= \{(\langle\rangle, ref') \mid okay \wedge tr' = tr \wedge wait'\} && [\text{Cases and SC}]\\
&\quad \cup \{(\langle\rangle, ref' \cup \{\checkmark\}) \mid okay \wedge wait' \wedge tr' = tr\}
\end{aligned}
$$

$$
\begin{array}{lr}
= \{(\langle\rangle, ref')\} \cup \{\} \cup \{(\langle\rangle, ref' \cup \{\checkmark\})\} \cup \{\} & [\text{ST}] \\
= \{(\langle\rangle, ref')\} \cup \{(\langle\rangle, ref' \cup \{\checkmark\})\} & [ref' \text{ definition}] \\
= \{(\langle\rangle, X) \mid X \subseteq \Sigma\} \cup \{(\langle\rangle, X \cup \{\checkmark\}) \mid X \subseteq \Sigma\} & [\text{ST}] \\
= \{(\langle\rangle, X) \mid X \subseteq \Sigma \cup \{\checkmark\}\} & [\Sigma^{\checkmark}] \\
= \{(\langle\rangle, X) \mid X \subseteq \Sigma^{\checkmark}\} & [failures] \\
= failures(\mathtt{STOP}) & [\Upsilon] \\
= failures(\Upsilon(Stop)) &
\end{array}
$$

Theorem J.23 $failures^{\mathcal{UTP}}(c \rightarrow Skip) = failures(\Upsilon(c \rightarrow Skip))$

Proof.

$$
\begin{array}{lr}
failures^{\mathcal{UTP}}(c \rightarrow Skip) & [failures^{\mathcal{UTP}}] \\
= \{(tr' - tr, ref') \mid (c \rightarrow Skip)^n\} & [A^t] \\
\quad \cup \{(tr' - tr, ref' \cup \{\checkmark\}) \mid (c \rightarrow Skip)^n \wedge wait'\} & \\
\quad \cup \{((tr' - tr) \frown \langle\checkmark\rangle, ref') \mid (c \rightarrow Skip)^t\} & \\
\quad \cup \{((tr' - tr) \frown \langle\checkmark\rangle, ref' \cup \{\checkmark\}) \mid (c \rightarrow Skip)^t\} & \\
= \{(tr' - tr, ref') \mid (c \rightarrow Skip)^n\} & [A^n] \\
\quad \cup \{(tr' - tr, ref' \cup \{\checkmark\}) \mid (c \rightarrow Skip)^n \wedge wait'\} & \\
\quad \cup \{((tr' - tr) \frown \langle\checkmark\rangle, ref') \mid \neg\, wait' \wedge (c \rightarrow Skip)^n\} & \\
\quad \cup \{((tr' - tr) \frown \langle\checkmark\rangle, ref' \cup \{\checkmark\}) \mid \neg\, wait' \wedge (c \rightarrow Skip)^n\} & \\
= \left\{ \begin{array}{c} (tr' - tr, ref') \mid okay \wedge \neg\, wait \wedge okay' \\ \wedge\ c \rightarrow Skip \end{array} \right\} & [\text{PC}] \\
\quad \cup \left\{ \begin{array}{c} (tr' - tr, ref' \cup \{\checkmark\}) \mid okay \wedge \neg\, wait \wedge okay' \wedge wait' \\ \wedge\ c \rightarrow Skip \end{array} \right\} & \\
\quad \cup \left\{ \begin{array}{c} ((tr' - tr) \frown \langle\checkmark\rangle, ref') \mid okay \wedge \neg\, wait \wedge okay' \wedge \neg\, wait' \\ \wedge\ c \rightarrow Skip \end{array} \right\} & \\
\quad \cup \left\{ \begin{array}{c} ((tr' - tr) \frown \langle\checkmark\rangle, ref' \cup \{\checkmark\}) \mid okay \wedge \neg\, wait \wedge okay' \wedge \neg\, wait' \\ \wedge\ c \rightarrow Skip \end{array} \right\} & \\
= \left\{ \begin{array}{c} (tr' - tr, ref') \mid okay \wedge \neg\, wait \wedge okay' \\ \wedge\ (c \rightarrow Skip)^t_f \end{array} \right\} & [\text{PC}] \\
\quad \cup \left\{ \begin{array}{c} (tr' - tr, ref' \cup \{\checkmark\}) \mid okay \wedge \neg\, wait \wedge okay' \wedge wait' \\ \wedge\ (c \rightarrow Skip)^t_f \end{array} \right\} & \\
\quad \cup \left\{ \begin{array}{c} ((tr' - tr) \frown \langle\checkmark\rangle, ref') \mid okay \wedge \neg\, wait \wedge okay' \wedge \neg\, wait' \\ \wedge\ (c \rightarrow Skip)^t_f \end{array} \right\} & \\
\quad \cup \left\{ \begin{array}{c} ((tr' - tr) \frown \langle\checkmark\rangle, ref' \cup \{\checkmark\}) \mid okay \wedge \neg\, wait \wedge okay' \wedge \neg\, wait' \\ \wedge\ (c \rightarrow Skip)^t_f \end{array} \right\} &
\end{array}
$$

$$
\begin{array}{ll}
= \left\{ \begin{array}{l} (tr' - tr, ref') \mid okay \wedge \neg\, wait \wedge okay' \\ \qquad \wedge\, ((c \to Skip)_f)^t \end{array} \right\} & \text{[Lemma J.3]} \\
\quad \cup \left\{ \begin{array}{l} (tr' - tr, ref' \cup \{\checkmark\}) \mid okay \wedge \neg\, wait \wedge okay' \wedge wait' \\ \qquad \wedge\, ((c \to Skip)_f)^t \end{array} \right\} & \\
\quad \cup \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref') \mid okay \wedge \neg\, wait \wedge okay' \wedge \neg\, wait' \\ \qquad \wedge\, ((c \to Skip)_f)^t \end{array} \right\} & \\
\quad \cup \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref' \cup \{\checkmark\}) \mid okay \wedge \neg\, wait \wedge okay' \wedge \neg\, wait' \\ \qquad \wedge\, ((c \to Skip)_f)^t \end{array} \right\} & \\
= \left\{ \begin{array}{l} (tr' - tr, ref') \mid okay \wedge \neg\, wait \wedge okay' \\ \qquad \wedge\, (\mathbf{CSP1}(okay' \wedge do_{\mathcal{C}}(c, Sync) \wedge v' = v))^t \end{array} \right\} & \text{[Lemma J.4]} \\
\quad \cup \left\{ \begin{array}{l} (tr' - tr, ref' \cup \{\checkmark\}) \mid okay \wedge \neg\, wait \wedge okay' \wedge wait' \\ \qquad \wedge\, (\mathbf{CSP1}(okay' \wedge do_{\mathcal{C}}(c, Sync) \wedge v' = v))^t \end{array} \right\} & \\
\quad \cup \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref') \mid okay \wedge \neg\, wait \wedge okay' \wedge \neg\, wait' \\ \qquad \wedge\, (\mathbf{CSP1}(okay' \wedge do_{\mathcal{C}}(c, Sync) \wedge v' = v))^t \end{array} \right\} & \\
\quad \cup \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref' \cup \{\checkmark\}) \mid okay \wedge \neg\, wait \wedge okay' \wedge \neg\, wait' \\ \qquad \wedge\, (\mathbf{CSP1}(okay' \wedge do_{\mathcal{C}}(c, Sync) \wedge v' = v))^t \end{array} \right\} & \\
= \left\{ \begin{array}{l} (tr' - tr, ref') \mid okay \wedge \neg\, wait \wedge okay' \\ \qquad \wedge\, (okay' \wedge do_{\mathcal{C}}(c, Sync) \wedge v' = v)^t \end{array} \right\} & \text{[PC]} \\
\quad \cup \left\{ \begin{array}{l} (tr' - tr, ref' \cup \{\checkmark\}) \mid okay \wedge \neg\, wait \wedge okay' \wedge wait' \\ \qquad \wedge\, (okay' \wedge do_{\mathcal{C}}(c, Sync) \wedge v' = v)^t \end{array} \right\} & \\
\quad \cup \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref') \mid okay \wedge \neg\, wait \wedge okay' \wedge \neg\, wait' \\ \qquad \wedge\, (okay' \wedge do_{\mathcal{C}}(c, Sync) \wedge v' = v)^t \end{array} \right\} & \\
\quad \cup \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref' \cup \{\checkmark\}) \mid okay \wedge \neg\, wait \wedge okay' \wedge \neg\, wait' \\ \qquad \wedge\, (okay' \wedge do_{\mathcal{C}}(c, Sync) \wedge v' = v)^t \end{array} \right\} & \\
= \left\{ \begin{array}{l} (tr' - tr, ref') \mid okay \wedge \neg\, wait \wedge okay' \\ \qquad \wedge\, do_{\mathcal{C}}(c, Sync) \wedge v' = v \end{array} \right\} & \text{[PC and } do_{\mathcal{C}}\text{]} \\
\quad \cup \left\{ \begin{array}{l} (tr' - tr, ref' \cup \{\checkmark\}) \mid okay \wedge \neg\, wait \wedge okay' \wedge wait' \\ \qquad \wedge\, do_{\mathcal{C}}(c, Sync) \wedge v' = v \end{array} \right\} & \\
\quad \cup \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref') \mid okay \wedge \neg\, wait \wedge okay' \wedge \neg\, wait' \\ \qquad \wedge\, do_{\mathcal{C}}(c, Sync) \wedge v' = v \end{array} \right\} & \\
\quad \cup \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref' \cup \{\checkmark\}) \mid okay \wedge \neg\, wait \wedge okay' \wedge \neg\, wait' \\ \qquad \wedge\, do_{\mathcal{C}}(c, Sync) \wedge v' = v \end{array} \right\} &
\end{array}
$$

$$= \left\{ \begin{array}{l} (tr' - tr, ref') \mid \\ \quad okay \land \neg\, wait \land okay' \land v' = v \\ \quad \land (tr' = tr \land (c, Sync) \notin ref' \lhd wait' \rhd tr' = tr ^\frown \langle (c, Sync) \rangle) \end{array} \right\} \quad [\text{PC}]$$

$$\cup \left\{ \begin{array}{l} (tr' - tr, ref' \cup \{\checkmark\}) \mid \\ \quad okay \land \neg\, wait \land okay' \land wait' \land v' = v \\ \quad \land (tr' = tr \land (c, Sync) \notin ref' \lhd wait' \rhd tr' = tr ^\frown \langle (c, Sync) \rangle) \end{array} \right\}$$

$$\cup \left\{ \begin{array}{l} ((tr' - tr) ^\frown \langle \checkmark \rangle, ref') \mid \\ \quad okay \land \neg\, wait \land okay' \land \neg\, wait' \land v' = v \\ \quad \land (tr' = tr \land (c, Sync) \notin ref' \lhd wait' \rhd tr' = tr ^\frown \langle (c, Sync) \rangle) \end{array} \right\}$$

$$\cup \left\{ \begin{array}{l} ((tr' - tr) ^\frown \langle \checkmark \rangle, ref' \cup \{\checkmark\}) \mid \\ \quad okay \land \neg\, wait \land okay' \land \neg\, wait' \land v' = v \\ \quad \land (tr' = tr \land (c, Sync) \notin ref' \lhd wait' \rhd tr' = tr ^\frown \langle (c, Sync) \rangle) \end{array} \right\}$$

$$= \left\{ \begin{array}{l} (tr' - tr, ref') \mid \\ \quad okay \land \neg\, wait \land okay' \land v' = v \land wait' \land tr' = tr \land (c, Sync) \notin ref' \end{array} \right\}$$

$$\cup \left\{ \begin{array}{l} (tr' - tr, ref') \mid \\ \quad okay \land \neg\, wait \land okay' \land v' = v \land \neg\, wait' \land tr' = tr ^\frown \langle (c, Sync) \rangle \end{array} \right\}$$

$$\cup \left\{ \begin{array}{l} (tr' - tr, ref' \cup \{\checkmark\}) \mid \\ \quad okay \land \neg\, wait \land okay' \land wait' \land v' = v \land tr' = tr \land (c, Sync) \notin ref' \end{array} \right\}$$

$$\cup \left\{ \begin{array}{l} ((tr' - tr) ^\frown \langle \checkmark \rangle, ref') \mid \\ \quad okay \land \neg\, wait \land okay' \land \neg\, wait' \land v' = v \land tr' = tr ^\frown \langle (c, Sync) \rangle \end{array} \right\}$$

$$\cup \left\{ \begin{array}{l} ((tr' - tr) ^\frown \langle \checkmark \rangle, ref' \cup \{\checkmark\}) \mid \\ \quad okay \land \neg\, wait \land okay' \land \neg\, wait' \land v' = v \land tr' = tr ^\frown \langle (c, Sync) \rangle \end{array} \right\}$$

$$[\text{SS. and } -]$$

$$= \left\{ \begin{array}{l} (\langle\rangle, ref') \mid \\ \quad okay \land \neg\, wait \land okay' \land v' = v \land wait' \land tr' = tr \land (c, Sync) \notin ref' \end{array} \right\}$$

$$\cup \left\{ \begin{array}{l} (\langle (c, Sync) \rangle, ref') \mid \\ \quad okay \land \neg\, wait \land okay' \land v' = v \land \neg\, wait' \land tr' = tr ^\frown \langle (c, Sync) \rangle \end{array} \right\}$$

$$\cup \left\{ \begin{array}{l} (\langle\rangle, ref' \cup \{\checkmark\}) \mid \\ \quad okay \land \neg\, wait \land okay' \land wait' \land v' = v \land tr' = tr \land (c, Sync) \notin ref' \end{array} \right\}$$

$$\cup \left\{ \begin{array}{l} (\langle (c, Sync), \checkmark \rangle, ref') \mid \\ \quad okay \land \neg\, wait \land okay' \land \neg\, wait' \land v' = v \land tr' = tr ^\frown \langle (c, Sync) \rangle \end{array} \right\}$$

$$\cup \left\{ \begin{array}{l} (\langle (c, Sync), \checkmark \rangle, ref' \cup \{\checkmark\}) \mid \\ \quad okay \land \neg\, wait \land okay' \land \neg\, wait' \land v' = v \land tr' = tr ^\frown \langle (c, Sync) \rangle \end{array} \right\}$$

$$[\textit{Circus}\ \text{Events}\ (c \equiv (c, Sync))]$$

$$
\begin{array}{l}
= \left\{ \begin{array}{l} (\langle\rangle, ref') \mid \\ \qquad okay \land \neg\, wait \land okay' \land v' = v \land wait' \land tr' = tr \land c \notin ref' \end{array} \right\} \\
\quad \cup \left\{ \begin{array}{l} (\langle c \rangle, ref') \mid \\ \qquad okay \land \neg\, wait \land okay' \land v' = v \land \neg\, wait' \land tr' = tr \frown \langle\, c \rangle \end{array} \right\} \\
\quad \cup \left\{ \begin{array}{l} (\langle\rangle, ref' \cup \{\checkmark\}) \mid \\ \qquad okay \land \neg\, wait \land okay' \land wait' \land v' = v \land tr' = tr \land c \notin ref' \end{array} \right\} \\
\quad \cup \left\{ \begin{array}{l} (\langle c, \checkmark \rangle, ref') \mid \\ \qquad okay \land \neg\, wait \land okay' \land \neg\, wait' \land v' = v \land tr' = tr \frown \langle\, c \rangle \end{array} \right\} \\
\quad \cup \left\{ \begin{array}{l} (\langle c, \checkmark \rangle, ref' \cup \{\checkmark\}) \mid \\ \qquad okay \land \neg\, wait \land okay' \land \neg\, wait' \land v' = v \land tr' = tr \frown \langle\, c \rangle \end{array} \right\}
\end{array}
$$

[Cases and SC]

$$
\begin{array}{ll}
= \{(\langle\rangle, ref') \mid c \notin ref'\} \cup \{\} & \text{[ST]} \\
\quad \cup \{(\langle c \rangle, ref')\} \cup \{\} & \\
\quad \cup \{(\langle\rangle, ref' \cup \{\checkmark\}) \mid c \notin ref'\} \cup \{\} & \\
\quad \cup \{(\langle c, \checkmark \rangle, ref')\} \cup \{\} & \\
\quad \cup \{(\langle c, \checkmark \rangle, ref' \cup \{\checkmark\})\} \cup \{\} & \\
= \{(\langle\rangle, ref') \mid c \notin ref'\} & [ref' \text{ definition}] \\
\quad \cup \{(\langle c \rangle, ref')\} & \\
\quad \cup \{(\langle\rangle, ref' \cup \{\checkmark\}) \mid c \notin ref'\} & \\
\quad \cup \{(\langle c, \checkmark \rangle, ref')\} & \\
\quad \cup \{(\langle c, \checkmark \rangle, ref' \cup \{\checkmark\})\} & \\
= \{(\langle\rangle, X) \mid X \subseteq \Sigma \land c \notin X\}\} & \text{[ST]} \\
\quad \cup \{(\langle c \rangle, X) \mid X \subseteq \Sigma\}\} & \\
\quad \cup \{(\langle\rangle, X \cup \{\checkmark\}) \mid X \subseteq \Sigma \land c \notin X\} & \\
\quad \cup \{(\langle c, \checkmark \rangle, X) \mid X \subseteq \Sigma\} & \\
\quad \cup \{(\langle c, \checkmark \rangle, X \cup \{\checkmark\}) \mid X \subseteq \Sigma\} & \\
= \{(\langle\rangle, X) \mid c \notin X \land X \subseteq \Sigma^{\checkmark}\} & [\frown] \\
\quad \cup \{(\langle c \rangle, X) \mid X \subseteq \Sigma\} & \\
\quad \cup \{(\langle c, \checkmark \rangle, X) \mid X \subseteq \Sigma^{\checkmark}\} & \\
= \{(\langle\rangle, X) \mid c \notin X \land X \subseteq \Sigma^{\checkmark}\} & \text{[SC]} \\
\quad \cup \{(\langle c \rangle \frown \langle\rangle, X) \mid X \subseteq \Sigma\} & \\
\quad \cup \{(\langle c \rangle \frown \langle \checkmark \rangle, X) \mid X \subseteq \Sigma^{\checkmark}\} & \\
= \{(\langle\rangle, X) \mid c \notin X \land X \subseteq \Sigma^{\checkmark}\} & \text{[ST]} \\
\quad \cup \{(\langle c \rangle \frown s, X) \mid (s, X) \in \{(\langle\rangle, X) \mid X \subseteq \Sigma\}\} & \\
\quad \cup \{(\langle c \rangle \frown s, X) \mid (s, X) \in \{(\langle \checkmark \rangle, X) \mid X \subseteq \Sigma^{\checkmark}\}\} & \\
= \{(\langle\rangle, X) \mid c \notin X \land X \subseteq \Sigma^{\checkmark}\} & [\mathit{failures}] \\
\quad \cup \{(\langle c \rangle \frown s, X) \mid (s, X) \in \{(\langle\rangle, X) \mid X \subseteq \Sigma\} \cup \{(\langle \checkmark \rangle, X) \mid X \subseteq \Sigma^{\checkmark}\}\} & \\
= \{(\langle\rangle, X) \mid c \notin X \land X \subseteq \Sigma^{\checkmark}\} \cup \{(\langle c \rangle \frown s, X) \mid (s, X) \in \mathit{failures}(\mathtt{SKIP})\} & [\mathit{failures}]
\end{array}
$$

$$= \mathit{failures}(\mathtt{c} \to \mathtt{SKIP}) \qquad [\Upsilon]$$
$$= \mathit{failures}(\Upsilon(c \to \mathit{Skip}))$$

Theorem J.24 $\mathit{failures}^{\mathcal{UTP}}(c \to A) = \mathit{failures}(\Upsilon(c \to A))$ *provided* $c \to A$ *is* $\boldsymbol{R}$

Inductive Hypothesis:

$$\mathit{failures}^{\mathcal{UTP}}(A) = \mathit{failures}(A)$$

Proof.

$$\begin{array}{ll}
\mathit{failures}^{\mathcal{UTP}}(c \to A) & [\mathit{failures}^{\mathcal{UTP}}] \\
= \{(tr' - tr, ref') \mid (c \to A)^n\} & [A^t] \\
\quad \cup \{(tr' - tr, ref' \cup \{\checkmark\}) \mid (c \to A)^n \wedge wait'\} & \\
\quad \cup \{((tr' - tr) \frown \langle\checkmark\rangle, ref') \mid (c \to A)^t\} & \\
\quad \cup \{((tr' - tr) \frown \langle\checkmark\rangle, ref' \cup \{\checkmark\}) \mid (c \to A)^t\} & \\
= \{(tr' - tr, ref') \mid (c \to A)^n\} & [A^n] \\
\quad \cup \{(tr' - tr, ref' \cup \{\checkmark\}) \mid (c \to A)^n \wedge wait'\} & \\
\quad \cup \{((tr' - tr) \frown \langle\checkmark\rangle, ref') \mid \neg\, wait' \wedge (c \to A)^n\} & \\
\quad \cup \{((tr' - tr) \frown \langle\checkmark\rangle, ref' \cup \{\checkmark\}) \mid \neg\, wait' \wedge (c \to A)^n\} & \\
= \left\{ \begin{array}{c} (tr' - tr, ref') \mid okay \wedge \neg\, wait \wedge okay' \\ \wedge\, c \to A \end{array} \right\} & [\text{PC}] \\
\quad \cup \left\{ \begin{array}{c} (tr' - tr, ref' \cup \{\checkmark\}) \mid okay \wedge \neg\, wait \wedge okay' \wedge wait' \\ \wedge\, c \to A \end{array} \right\} & \\
\quad \cup \left\{ \begin{array}{c} ((tr' - tr) \frown \langle\checkmark\rangle, ref') \mid okay \wedge \neg\, wait \wedge okay' \wedge \neg\, wait' \\ \wedge\, c \to A \end{array} \right\} & \\
\quad \cup \left\{ \begin{array}{c} ((tr' - tr) \frown \langle\checkmark\rangle, ref' \cup \{\checkmark\}) \mid okay \wedge \neg\, wait \wedge okay' \wedge \neg\, wait' \\ \wedge\, c \to A \end{array} \right\} & \\
= \left\{ \begin{array}{c} (tr' - tr, ref') \mid okay \wedge \\ \wedge\, (c \to A)^t_f \end{array} \right\} & [\text{Lemma J.6}] \\
\quad \cup \left\{ \begin{array}{c} (tr' - tr, ref' \cup \{\checkmark\}) \mid okay \wedge wait' \\ \wedge\, (c \to A)^t_f \end{array} \right\} & \\
\quad \cup \left\{ \begin{array}{c} ((tr' - tr) \frown \langle\checkmark\rangle, ref') \mid okay \wedge \neg\, wait' \\ \wedge\, (c \to A)^t_f \end{array} \right\} & \\
\quad \cup \left\{ \begin{array}{c} ((tr' - tr) \frown \langle\checkmark\rangle, ref' \cup \{\checkmark\}) \mid okay \wedge \neg\, wait' \\ \wedge\, (c \to A)^t_f \end{array} \right\} &
\end{array}$$

$$= \left\{ \begin{array}{l} (tr' - tr, ref') \\ \mid okay \land \\ \quad \land (\mathbf{CSP1}(okay' \land do_{\mathcal{C}}(c, Sync) \land v' = v);\ (A)^t) \end{array} \right\} \qquad [\text{Lemma J.5}]$$

$$\cup \left\{ \begin{array}{l} (tr' - tr, ref' \cup \{\checkmark\}) \\ \mid okay \land wait' \\ \quad \land (\mathbf{CSP1}(okay' \land do_{\mathcal{C}}(c, Sync) \land v' = v);\ (A)^t) \end{array} \right\}$$

$$\cup \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref') \\ \mid okay \land \neg\, wait' \\ \quad \land (\mathbf{CSP1}(okay' \land do_{\mathcal{C}}(c, Sync) \land v' = v);\ (A)^t) \end{array} \right\}$$

$$\cup \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref' \cup \{\checkmark\}) \\ \mid okay \land \neg\, wait' \\ \quad \land (\mathbf{CSP1}(okay' \land do_{\mathcal{C}}(c, Sync) \land v' = v);\ (A)^t) \end{array} \right\}$$

$$= \left\{ \begin{array}{l} (tr' - tr, ref') \\ \mid okay \land \\ \quad \land ((okay' \land do_{\mathcal{C}}(c, Sync) \land v' = v);\ (A)^t) \end{array} \right\} \qquad [do_{\mathcal{C}}]$$

$$\cup \left\{ \begin{array}{l} (tr' - tr, ref' \cup \{\checkmark\}) \\ \mid okay \land wait' \\ \quad \land ((okay' \land do_{\mathcal{C}}(c, Sync) \land v' = v);\ (A)^t) \end{array} \right\}$$

$$\cup \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref') \\ \mid okay \land \neg\, wait' \\ \quad \land ((okay' \land do_{\mathcal{C}}(c, Sync) \land v' = v);\ (A)^t) \end{array} \right\}$$

$$\cup \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref' \cup \{\checkmark\}) \\ \mid okay \land \neg\, wait' \\ \quad \land ((okay' \land do_{\mathcal{C}}(c, Sync) \land v' = v);\ (A)^t) \end{array} \right\}$$

$$
\begin{array}{ll}
= \left\{ \begin{array}{l} (tr' - tr, ref') \\ \mid okay \land \left(\left(\begin{array}{l} okay' \land v' = v \\ \land \left(\begin{array}{l} tr' = tr \land (c, Sync) \notin ref' \\ \lhd wait' \rhd \\ tr' = tr \frown \langle (c, Sync) \rangle \end{array} \right) \end{array} \right) ; \\ (A)^t \end{array} \right) \end{array} \right\} & [PC] \\
\cup \left\{ \begin{array}{l} (tr' - tr, ref' \cup \{\checkmark\}) \\ \mid okay \land wait' \land \left(\left(\begin{array}{l} okay' \land v' = v \\ \land \left(\begin{array}{l} tr' = tr \land (c, Sync) \notin ref' \\ \lhd wait' \rhd \\ tr' = tr \frown \langle (c, Sync) \rangle \end{array} \right) \end{array} \right) ; \\ (A)^t \end{array} \right) \end{array} \right\} & \\
\cup \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref') \\ \mid okay \land \neg\, wait' \land \left(\left(\begin{array}{l} okay' \land v' = v \\ \land \left(\begin{array}{l} tr' = tr \land (c, Sync) \notin ref' \\ \lhd wait' \rhd \\ tr' = tr \frown \langle (c, Sync) \rangle \end{array} \right) \end{array} \right) ; \\ (A)^t \end{array} \right) \end{array} \right\} & \\
\cup \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref' \cup \{\checkmark\}) \\ \mid okay \land \neg\, wait' \land \left(\left(\begin{array}{l} okay' \land v' = v \\ \land \left(\begin{array}{l} tr' = tr \land (c, Sync) \notin ref' \\ \lhd wait' \rhd \\ tr' = tr \frown \langle (c, Sync) \rangle \end{array} \right) \end{array} \right) ; \\ (A)^t \end{array} \right) \end{array} \right\} &
\end{array}
$$

$$
\begin{aligned}
= & \left\{ \begin{array}{l} (tr' - tr, ref') \\ \mid \left(okay \wedge \left(\begin{array}{l} \left(\begin{array}{l} okay' \wedge v' = v \wedge wait' \\ \wedge\, tr' = tr \wedge (c, Sync) \notin ref' \end{array} \right); \\ (A)^t \end{array} \right) \right) \\ \vee \left(okay \wedge \left(\begin{array}{l} \left(\begin{array}{l} okay' \wedge v' = v \wedge \neg\, wait' \\ \wedge\, tr' = tr \frown \langle (c, Sync) \rangle \end{array} \right); \\ (A)^t \end{array} \right) \right) \end{array} \right\} \\
\cup & \left\{ \begin{array}{l} (tr' - tr, ref' \cup \{\checkmark\}) \\ \mid \left(okay \wedge wait' \wedge \left(\begin{array}{l} \left(\begin{array}{l} okay' \wedge v' = v \wedge wait' \\ \wedge\, tr' = tr \wedge (c, Sync) \notin ref' \end{array} \right); \\ (A)^t \end{array} \right) \right) \\ \vee \left(okay \wedge wait' \wedge \left(\begin{array}{l} \left(\begin{array}{l} okay' \wedge v' = v \wedge \neg\, wait' \\ \wedge\, tr' = tr \frown \langle (c, Sync) \rangle \end{array} \right); \\ (A)^t \end{array} \right) \right) \end{array} \right\} \\
\cup & \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref') \\ \mid \left(okay \wedge \neg\, wait' \wedge \left(\begin{array}{l} \left(\begin{array}{l} okay' \wedge v' = v \wedge wait' \\ \wedge\, tr' = tr \wedge (c, Sync) \notin ref' \end{array} \right); \\ (A)^t \end{array} \right) \right) \\ \vee \left(okay \wedge \neg\, wait' \wedge \left(\begin{array}{l} \left(\begin{array}{l} okay' \wedge v' = v \wedge \neg\, wait' \\ \wedge\, tr' = tr \frown \langle (c, Sync) \rangle \end{array} \right); \\ (A)^t \end{array} \right) \right) \end{array} \right\} \\
\cup & \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref' \cup \{\checkmark\}) \\ \mid \left(okay \wedge \neg\, wait' \wedge \left(\begin{array}{l} \left(\begin{array}{l} okay' \wedge v' = v \wedge wait' \\ \wedge\, tr' = tr \wedge (c, Sync) \notin ref' \end{array} \right); \\ (A)^t \end{array} \right) \right) \\ \vee \left(okay \wedge \neg\, wait' \wedge \left(\begin{array}{l} \left(\begin{array}{l} okay' \wedge v' = v \wedge \neg\, wait' \\ \wedge\, tr' = tr \frown \langle (c, Sync) \rangle \end{array} \right); \\ (A)^t \end{array} \right) \right) \end{array} \right\}
\end{aligned}
$$

[Lemma J.25 (proviso)]

$$= \left\{ \begin{array}{l} (tr' - tr, ref') \\ \mid (okay \wedge v' = v \wedge wait' \wedge tr' = tr \wedge (c, Sync) \notin ref') \\ \vee \left(okay \wedge \left(\begin{array}{l} \left(\begin{array}{l} okay' \wedge v' = v \wedge \neg\, wait' \\ \wedge\, tr' = tr \frown \langle\, (c, Sync) \rangle \end{array} \right); \\ (A)^t \end{array} \right) \right) \end{array} \right\}$$

$$\cup \left\{ \begin{array}{l} (tr' - tr, ref' \cup \{\checkmark\}) \\ \mid (okay \wedge wait' \wedge v' = v \wedge wait' \wedge tr' = tr \wedge (c, Sync) \notin ref') \\ \vee \left(okay \wedge wait' \wedge \left(\begin{array}{l} \left(\begin{array}{l} okay' \wedge v' = v \wedge \neg\, wait' \\ \wedge\, tr' = tr \frown \langle\, (c, Sync) \rangle \end{array} \right); \\ (A)^t \end{array} \right) \right) \end{array} \right\}$$

$$\cup \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref') \\ \mid (okay \wedge \neg\, wait' \wedge v' = v \wedge wait' \wedge tr' = tr \wedge (c, Sync) \notin ref') \\ \vee \left(okay \wedge \neg\, wait' \wedge \left(\begin{array}{l} \left(\begin{array}{l} okay' \wedge v' = v \wedge \neg\, wait' \\ \wedge\, tr' = tr \frown \langle\, (c, Sync) \rangle \end{array} \right); \\ (A)^t \end{array} \right) \right) \end{array} \right\}$$

$$\cup \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref' \cup \{\checkmark\}) \\ \mid (okay \wedge \neg\, wait' \wedge v' = v \wedge wait' \wedge tr' = tr \wedge (c, Sync) \notin ref') \\ \vee \left(okay \wedge \neg\, wait' \wedge \left(\begin{array}{l} \left(\begin{array}{l} okay' \wedge v' = v \wedge \neg\, wait' \\ \wedge\, tr' = tr \frown \langle\, (c, Sync) \rangle \end{array} \right); \\ (A)^t \end{array} \right) \right) \end{array} \right\}$$

[PC and SC]

$$= \{(tr' - tr, ref') \mid (okay \wedge v' = v \wedge wait' \wedge tr' = tr \wedge (c, Sync) \notin ref')\}$$

$$\cup \left\{ \begin{array}{l} (tr' - tr, ref') \\ \mid okay \wedge \left(\begin{array}{l} \left(\begin{array}{l} okay' \wedge v' = v \wedge \neg\, wait' \\ \wedge\, tr' = tr \frown \langle\, (c, Sync) \rangle \end{array} \right); \\ (A)^t \end{array} \right) \end{array} \right\}$$

$$\cup \{(tr' - tr, ref' \cup \{\checkmark\}) \mid (okay \wedge wait' \wedge v' = v \wedge tr' = tr \wedge (c, Sync) \notin ref')\}$$

$$\cup \left\{ \begin{array}{l} (tr' - tr, ref' \cup \{\checkmark\}) \\ \mid okay \wedge wait' \wedge \left(\begin{array}{l} \left(\begin{array}{l} okay' \wedge v' = v \wedge \neg\, wait' \\ \wedge\, tr' = tr \frown \langle\, (c, Sync) \rangle \end{array} \right); \\ (A)^t \end{array} \right) \end{array} \right\}$$

$$\cup \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref') \\ \mid \left(okay \wedge \neg\, wait' \wedge \left(\begin{array}{l} \left(\begin{array}{l} okay' \wedge v' = v \wedge \neg\, wait' \\ \wedge\, tr' = tr \frown \langle\, (c, Sync) \rangle \end{array} \right); \\ (A)^t \end{array} \right) \right) \end{array} \right\}$$

$$\cup \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref' \cup \{\checkmark\}) \\ \mid \left(okay \wedge \neg\, wait' \wedge \left(\begin{array}{l} \left(\begin{array}{l} okay' \wedge v' = v \wedge \neg\, wait' \\ \wedge\, tr' = tr \frown \langle\, (c, Sync) \rangle \end{array} \right); \\ (A)^t \end{array} \right) \right) \end{array} \right\}$$

[Lemma J.16]

$$
\begin{aligned}
&= \{(tr' - tr, ref') \mid (okay \land v' = v \land wait' \land tr' = tr \land (c, Sync) \notin ref')\} && \text{[ST]}\\
&\quad \cup \{(tr' - tr, ref') \mid okay \land (A)^t_f[tr \frown \langle (c, Sync)\rangle / tr]\}\\
&\quad \cup \{(tr' - tr, ref' \cup \{\checkmark\}) \mid (okay \land wait' \land v' = v \land tr' = tr \land (c, Sync) \notin ref')\}\\
&\quad \cup \{(tr' - tr, ref' \cup \{\checkmark\}) \mid okay \land wait' \land (A)^t_f[tr \frown \langle (c, Sync)\rangle / tr]\}\\
&\quad \cup \{((tr' - tr) \frown \langle \checkmark \rangle, ref') \mid okay \land \neg\, wait' \land (A)^t_f[tr \frown \langle (c, Sync)\rangle / tr]\}\\
&\quad \cup \{((tr' - tr) \frown \langle \checkmark \rangle, ref' \cup \{\checkmark\}) \mid okay \land \neg\, wait' \land (A)^t_f[tr \frown \langle (c, Sync)\rangle / tr]\}\\
&= \{(tr' - tr, ref') \mid (okay \land v' = v \land wait' \land tr' = tr \land (c, Sync) \notin ref')\}\\
&\quad \cup \{(tr' - tr, ref' \cup \{\checkmark\}) \mid (okay \land wait' \land v' = v \land tr' = tr \land (c, Sync) \notin ref')\}\\
&\quad \cup \{(tr' - tr, ref') \mid okay \land (A)^t_f[tr \frown \langle (c, Sync)\rangle / tr]\}\\
&\quad \cup \{(tr' - tr, ref' \cup \{\checkmark\}) \mid okay \land wait' \land (A)^t_f[tr \frown \langle (c, Sync)\rangle / tr]\}\\
&\quad \cup \{((tr' - tr) \frown \langle \checkmark \rangle, ref') \mid okay \land \neg\, wait' \land (A)^t_f[tr \frown \langle (c, Sync)\rangle / tr]\}\\
&\quad \cup \{((tr' - tr) \frown \langle \checkmark \rangle, ref' \cup \{\checkmark\}) \mid okay \land \neg\, wait' \land (A)^t_f[tr \frown \langle (c, Sync)\rangle / tr]\}\\
& && \text{[\textit{Circus} Events } (c \equiv (c, Sync))]\\
&= \{(tr' - tr, ref') \mid (okay \land v' = v \land wait' \land tr' = tr \land c \notin ref')\} && \text{[Lemma J.20 (proviso)]}\\
&\quad \cup \{(tr' - tr, ref' \cup \{\checkmark\}) \mid (okay \land wait' \land v' = v \land tr' = tr \land c \notin ref')\}\\
&\quad \cup \{(tr' - tr, ref') \mid okay \land (A)^t_f[tr \frown \langle c\rangle / tr]\}\\
&\quad \cup \{(tr' - tr, ref' \cup \{\checkmark\}) \mid okay \land wait' \land (A)^t_f[tr \frown \langle c\rangle / tr]\}\\
&\quad \cup \{((tr' - tr) \frown \langle \checkmark \rangle, ref') \mid okay \land \neg\, wait' \land (A)^t_f[tr \frown \langle c\rangle / tr]\}\\
&\quad \cup \{((tr' - tr) \frown \langle \checkmark \rangle, ref' \cup \{\checkmark\}) \mid okay \land \neg\, wait' \land (A)^t_f[tr \frown \langle c\rangle / tr]\}\\
&= \{(tr' - tr, ref') \mid (okay \land v' = v \land wait' \land tr' = tr \land c \notin ref')\} && \text{[SS and } -]\\
&\quad \cup \{(tr' - tr, ref' \cup \{\checkmark\}) \mid (okay \land wait' \land v' = v \land tr' = tr \land c \notin ref')\}\\
&\quad \cup \{(c \frown (tr' - tr), ref') \mid okay \land (A)^t_f\}\\
&\quad \cup \{((c \frown (tr' - tr), ref' \cup \{\checkmark\}) \mid okay \land wait' \land (A)^t_f\}\\
&\quad \cup \{((c \frown (tr' - tr) \frown \langle \checkmark \rangle, ref') \mid okay \land \neg\, wait' \land (A)^t_f\}\\
&\quad \cup \{((c \frown (tr' - tr) \frown \langle \checkmark \rangle, ref' \cup \{\checkmark\}) \mid okay \land \neg\, wait' \land (A)^t_f\}\\
&= \{(\langle\rangle, ref') \mid (okay \land v' = v \land wait' \land tr' = tr \land c \notin ref')\} && \text{[Cases and SC]}\\
&\quad \cup \{(\langle\rangle, ref' \cup \{\checkmark\}) \mid (okay \land wait' \land v' = v \land tr' = tr \land c \notin ref')\}\\
&\quad \cup \{(c \frown (tr' - tr), ref') \mid okay \land (A)^t_f\}\\
&\quad \cup \{((c \frown (tr' - tr), ref' \cup \{\checkmark\}) \mid okay \land wait' \land (A)^t_f\}\\
&\quad \cup \{((c \frown (tr' - tr) \frown \langle \checkmark \rangle, ref') \mid okay \land \neg\, wait' \land (A)^t_f\}\\
&\quad \cup \{((c \frown (tr' - tr) \frown \langle \checkmark \rangle, ref' \cup \{\checkmark\}) \mid okay \land \neg\, wait' \land (A)^t_f\}\\
&= \{(\langle\rangle, ref') \mid c \notin ref'\} \cup \{\} && \text{[ST]}\\
&\quad \cup \{(\langle\rangle, ref' \cup \{\checkmark\}) \mid c \notin ref'\} \cup \{\}\\
&\quad \cup \{(c \frown (tr' - tr), ref') \mid okay \land (A)^t_f\}\\
&\quad \cup \{((c \frown (tr' - tr), ref' \cup \{\checkmark\}) \mid okay \land wait' \land (A)^t_f\}\\
&\quad \cup \{((c \frown (tr' - tr) \frown \langle \checkmark \rangle, ref') \mid okay \land \neg\, wait' \land (A)^t_f\}\\
&\quad \cup \{((c \frown (tr' - tr) \frown \langle \checkmark \rangle, ref' \cup \{\checkmark\}) \mid okay \land \neg\, wait' \land (A)^t_f\}
\end{aligned}
$$

$$
\begin{aligned}
&= \{(\langle\rangle, \mathit{ref}') \mid c \notin \mathit{ref}'\} && [\mathit{ref}' \text{ definition}]\\
&\quad \cup \{(\langle\rangle, \mathit{ref}' \cup \{\checkmark\}) \mid c \notin \mathit{ref}'\}\\
&\quad \cup \{(c \frown (\mathit{tr}' - \mathit{tr}), \mathit{ref}') \mid \mathit{okay} \wedge (A)^t_f\}\\
&\quad \cup \{((c \frown (\mathit{tr}' - \mathit{tr}), \mathit{ref}' \cup \{\checkmark\}) \mid \mathit{okay} \wedge \mathit{wait}' \wedge (A)^t_f\}\\
&\quad \cup \{((c \frown (\mathit{tr}' - \mathit{tr}) \frown \langle\checkmark\rangle, \mathit{ref}') \mid \mathit{okay} \wedge \neg\, \mathit{wait}' \wedge (A)^t_f\}\\
&\quad \cup \{((c \frown (\mathit{tr}' - \mathit{tr}) \frown \langle\checkmark\rangle, \mathit{ref}' \cup \{\checkmark\}) \mid \mathit{okay} \wedge \neg\, \mathit{wait}' \wedge (A)^t_f\}\\
&= \{(\langle\rangle, \mathit{ref}') \mid c \notin \mathit{ref}' \wedge \mathit{ref}' \subseteq \Sigma\} && [\text{ST}]\\
&\quad \cup \{(\langle\rangle, \mathit{ref}' \cup \{\checkmark\}) \mid c \notin \mathit{ref}' \wedge \mathit{ref}' \subseteq \Sigma\}\\
&\quad \cup \{(c \frown (\mathit{tr}' - \mathit{tr}), \mathit{ref}') \mid \mathit{okay} \wedge (A)^t_f\}\\
&\quad \cup \{((c \frown (\mathit{tr}' - \mathit{tr}), \mathit{ref}' \cup \{\checkmark\}) \mid \mathit{okay} \wedge \mathit{wait}' \wedge (A)^t_f\}\\
&\quad \cup \{((c \frown (\mathit{tr}' - \mathit{tr}) \frown \langle\checkmark\rangle, \mathit{ref}') \mid \mathit{okay} \wedge \neg\, \mathit{wait}' \wedge (A)^t_f\}\\
&\quad \cup \{((c \frown (\mathit{tr}' - \mathit{tr}) \frown \langle\checkmark\rangle, \mathit{ref}' \cup \{\checkmark\}) \mid \mathit{okay} \wedge \neg\, \mathit{wait}' \wedge (A)^t_f\}\\
&= \{(\langle\rangle, \mathit{ref}') \mid c \notin \mathit{ref}' \wedge \mathit{ref}' \subseteq \Sigma^{\checkmark}\} && [\text{Rename } \mathit{ref}']\\
&\quad \cup \{(\langle c\rangle \frown (\mathit{tr}' - \mathit{tr}), \mathit{ref}') \mid \mathit{okay} \wedge (A)^t_f\}\\
&\quad \cup \{(\langle c\rangle \frown (\mathit{tr}' - \mathit{tr}), \mathit{ref}' \cup \{\checkmark\}) \mid \mathit{okay} \wedge \mathit{wait}' \wedge (A)^t_f\}\\
&\quad \cup \{(\langle c\rangle \frown (\mathit{tr}' - \mathit{tr}) \frown \langle\checkmark\rangle, \mathit{ref}') \mid \mathit{okay} \wedge \neg\, \mathit{wait}' \wedge (A)^t_f\}\\
&\quad \cup \{(\langle c\rangle \frown (\mathit{tr}' - \mathit{tr}) \frown \langle\checkmark\rangle, \mathit{ref}' \cup \{\checkmark\}) \mid \mathit{okay} \wedge \neg\, \mathit{wait}' \wedge (A)^t_f\}\\
&= \{(\langle\rangle, X) \mid c \notin X \wedge X \subseteq \Sigma^{\checkmark}\} && [\text{PC}]\\
&\quad \cup \{(\langle c\rangle \frown (\mathit{tr}' - \mathit{tr}), \mathit{ref}') \mid \mathit{okay} \wedge (A)^t_f\}\\
&\quad \cup \{(\langle c\rangle \frown (\mathit{tr}' - \mathit{tr}), \mathit{ref}' \cup \{\checkmark\}) \mid \mathit{okay} \wedge \mathit{wait}' \wedge (A)^t_f\}\\
&\quad \cup \{(\langle c\rangle \frown (\mathit{tr}' - \mathit{tr}) \frown \langle\checkmark\rangle, \mathit{ref}') \mid \mathit{okay} \wedge \neg\, \mathit{wait}' \wedge (A)^t_f\}\\
&\quad \cup \{(\langle c\rangle \frown (\mathit{tr}' - \mathit{tr}) \frown \langle\checkmark\rangle, \mathit{ref}' \cup \{\checkmark\}) \mid \mathit{okay} \wedge \neg\, \mathit{wait}' \wedge (A)^t_f\}\\
&= \{(\langle\rangle, X) \mid c \notin X \wedge X \subseteq \Sigma^{\checkmark}\} && [\text{SC}]\\
&\quad \cup \left\{\begin{array}{l} (\langle c\rangle \frown (\mathit{tr}' - \mathit{tr}), \mathit{ref}')\\ \mid \mathit{okay} \wedge \neg\, \mathit{wait} \wedge \mathit{okay}' \wedge A \end{array}\right\}\\
&\quad \cup \left\{\begin{array}{l} (\langle c\rangle \frown (\mathit{tr}' - \mathit{tr}), \mathit{ref}' \cup \{\checkmark\})\\ \mid \mathit{okay} \wedge \neg\, \mathit{wait} \wedge \mathit{okay}' \wedge A \wedge \mathit{wait}' \end{array}\right\}\\
&\quad \cup \left\{\begin{array}{l} (\langle c\rangle \frown (\mathit{tr}' - \mathit{tr}) \frown \langle\checkmark\rangle, \mathit{ref}')\\ \mid \mathit{okay} \wedge \neg\, \mathit{wait} \wedge \mathit{okay}' \wedge \neg\, \mathit{wait}' \wedge A \end{array}\right\}\\
&\quad \cup \left\{\begin{array}{l} (\langle c\rangle \frown (\mathit{tr}' - \mathit{tr}) \frown \langle\checkmark\rangle, \mathit{ref}' \cup \{\checkmark\})\\ \mid \mathit{okay} \wedge \neg\, \mathit{wait} \wedge \mathit{okay}' \wedge \neg\, \mathit{wait}' \wedge A \end{array}\right\}
\end{aligned}
$$

$$
\begin{array}{ll}
= \{(\langle\rangle, X) \mid c \notin X \wedge X \subseteq \Sigma^{\checkmark}\} & [A^n] \\
\cup \left\{ \begin{array}{l} (\langle c \rangle \frown s, X) \\ \mid (s, X) \in \left\{ \begin{array}{l} (tr' - tr, ref') \\ \mid okay \wedge \neg\, wait \wedge okay' \wedge A \end{array} \right\} \end{array} \right\} & \\
\cup \left\{ \begin{array}{l} (\langle c \rangle \frown s, X) \\ \mid (s, X) \in \left\{ \begin{array}{l} (tr' - tr, ref' \cup \{\checkmark\}) \\ \mid okay \wedge \neg\, wait \wedge okay' \wedge A \wedge wait' \end{array} \right\} \end{array} \right\} & \\
\cup \left\{ \begin{array}{l} (\langle c \rangle \frown s, X) \\ \mid (s, X) \in \left\{ \begin{array}{l} ((tr' - tr) \frown \langle\checkmark\rangle, ref') \\ \mid okay \wedge \neg\, wait \wedge okay' \wedge \neg\, wait' \wedge A \end{array} \right\} \end{array} \right\} & \\
\cup \left\{ \begin{array}{l} (\langle c \rangle \frown s, X) \\ \mid (s, X) \in \left\{ \begin{array}{l} ((tr' - tr) \frown \langle\checkmark\rangle, ref' \cup \{\checkmark\}) \\ \mid okay \wedge \neg\, wait \wedge okay' \wedge \neg\, wait' \wedge A \end{array} \right\} \end{array} \right\} & \\
= \{(\langle\rangle, X) \mid c \notin X \wedge X \subseteq \Sigma^{\checkmark}\} & [A^t] \\
\quad \cup \{(\langle c \rangle \frown s, X) \mid (s, X) \in \{(tr' - tr, ref') \mid (A)^n\}\} & \\
\quad \cup \{(\langle c \rangle \frown s, X) \mid (s, X) \in \{(tr' - tr, ref' \cup \{\checkmark\}) \mid (A)^n \wedge wait'\}\} & \\
\quad \cup \{(\langle c \rangle \frown s, X) \mid (s, X) \in \{((tr' - tr) \frown \langle\checkmark\rangle, ref') \mid \neg\, wait' \wedge (A)^n\}\} & \\
\quad \cup \{(\langle c \rangle \frown s, X) \mid (s, X) \in \{((tr' - tr) \frown \langle\checkmark\rangle, ref' \cup \{\checkmark\}) \mid \neg\, wait' \wedge (A)^n\}\} & \\
= \{(\langle\rangle, X) \mid c \notin X \wedge X \subseteq \Sigma^{\checkmark}\} & [\text{SC e ST}] \\
\quad \cup \{(\langle c \rangle \frown s, X) \mid (s, X) \in \{(tr' - tr, ref') \mid (A)^n\}\} & \\
\quad \cup \{(\langle c \rangle \frown s, X) \mid (s, X) \in \{(tr' - tr, ref' \cup \{\checkmark\}) \mid (A)^n \wedge wait'\}\} & \\
\quad \cup \{(\langle c \rangle \frown s, X) \mid (s, X) \in \{((tr' - tr) \frown \langle\checkmark\rangle, ref') \mid (A)^t\}\} & \\
\quad \cup \{(\langle c \rangle \frown s, X) \mid (s, X) \in \{((tr' - tr) \frown \langle\checkmark\rangle, ref' \cup \{\checkmark\}) \mid (A)^t\}\} & \\
= \{(\langle\rangle, X) \mid c \notin X \wedge X \subseteq \Sigma^{\checkmark}\} & [failures^{\mathcal{UTP}}] \\
\cup \left\{ \begin{array}{l} (\langle c \rangle \frown s, X) \\ \mid (s, X) \in \left\{ \begin{array}{l} \{(tr' - tr, ref') \mid (A)^n\} \\ \cup \{(tr' - tr, ref' \cup \{\checkmark\}) \mid (A)^n \wedge wait'\} \\ \cup \{((tr' - tr) \frown \langle\checkmark\rangle, ref') \mid (A)^t\} \\ \cup \{((tr' - tr) \frown \langle\checkmark\rangle, ref' \cup \{\checkmark\}) \mid (A)^t\} \end{array} \right\} \end{array} \right\} & \\
= \{(\langle\rangle, X) \mid c \notin X \wedge X \subseteq \Sigma^{\checkmark}\} \cup \{(\langle c \rangle \frown s, X) \mid (s, X) \in failures^{\mathcal{UTP}}(A)\} & [\text{IH}] \\
= \{(\langle\rangle, X) \mid c \notin X \wedge X \subseteq \Sigma^{\checkmark}\} \cup \{(\langle c \rangle \frown s, X) \mid (s, X) \in failures(\Upsilon(A))\} & [failures] \\
= failures(\mathsf{c} \rightarrow \Upsilon(A)) & [\Upsilon] \\
= failures(\Upsilon(c \rightarrow A)) &
\end{array}
$$

Theorem J.25 $failures^{\mathcal{UTP}}(c.v \rightarrow A) = failures(\Upsilon(c.v \rightarrow A))$
provided A is $\boldsymbol{R}$

Proof. Identical to that of Theorem J.24, but replacing $Sync$ by v.

Theorem J.26 $\mathit{failures}^{\mathcal{UTP}}(c!v \rightarrow A) = \mathit{failures}(\Upsilon(c.v \rightarrow A))$
provided A *is* $\boldsymbol{R}$

Proof. Using the ***Circus*** semantics of $c!v \rightarrow A \equiv c.v \rightarrow A$ and Theorem J.25.

Theorem J.27

$$\mathit{failures}^{\mathcal{UTP}}(c?x : P \rightarrow A) = \mathit{failures}(\Upsilon(c?x : P \rightarrow A))$$

provided

1. $c?x : P \rightarrow A$ *is* $\boldsymbol{R}$
2. $c?x : P \rightarrow A$ *is divergence-free*

Inductive Hypothesis (A):

$$\forall v : S \bullet \mathit{failures}^{\mathcal{UTP}}(A[v/x])) = \mathit{failures}(\Upsilon(A)[v/x])$$

Proof.

$$\begin{array}{lr}
\mathit{failures}^{\mathcal{UTP}}(c?x : P \rightarrow A) & [\text{Property of } \textit{Circus} \text{ input}] \\
= \mathit{failures}^{\mathcal{UTP}}(\Box\, v : \{x : \delta(c) \mid P\} \bullet c.v \rightarrow A[v/x]) & \\
& [\text{Theorems J.30 and J.25 (IH)}] \\
= \mathit{failures}(\Upsilon(\Box\, v : \{x : \delta(c) \mid P\} \bullet c.v \rightarrow A[v/x])) & \\
& [\text{Property of } \textit{Circus} \text{ input}] \\
= \mathit{failures}(\Upsilon(c?x : P \rightarrow A)) &
\end{array}$$

Theorem J.28

$$\mathit{failures}^{\mathcal{UTP}}(c?x \rightarrow A) = \mathit{failures}(\Upsilon(c?x \rightarrow A))$$

provided

1. $c?x : P \rightarrow A$ *is* $\boldsymbol{R}$
2. $c?x : P \rightarrow A$ *is divergence-free*
3. $v : S \bullet \mathit{failures}^{\mathcal{UTP}}(A[v/x])) = \mathit{failures}(\Upsilon(A)[v/x])$

Proof. Using the *Circus* semantics of $c?x \rightarrow A \equiv c?x : true \rightarrow A$ and Theorem J.27.

Theorem J.29 $\mathit{failures}^{\mathcal{UTP}}(A \Box B) = \mathit{failures}(\Upsilon(A \Box B))$
provided

1. *A and B are* $\boldsymbol{R}$
2. *A and B are divergence-free*

Inductive Hypothesis:

$$\mathit{failures}^{\mathcal{UTP}}(A) = \mathit{failures}(\Upsilon(A))$$
$$\mathit{failures}^{\mathcal{UTP}}(B) = \mathit{failures}(\Upsilon(B))$$

Proof.

$$\begin{array}{ll}
\mathit{failures}^{\mathcal{UTP}}(A \Box B) & [\mathit{failures}^{\mathcal{UTP}}] \\
= \{(tr' - tr, ref') \mid (A \Box B)^n\} & [A^t] \\
\quad \cup \{(tr' - tr, ref' \cup \{\checkmark\}) \mid (A \Box B)^n \land wait'\} & \\
\quad \cup \{((tr' - tr) \frown \langle \checkmark \rangle, ref') \mid (A \Box B)^t\} & \\
\quad \cup \{((tr' - tr) \frown \langle \checkmark \rangle, ref' \cup \{\checkmark\}) \mid (A \Box B)^t\} & \\
= \{(tr' - tr, ref') \mid (A \Box B)^n\} & [A^n] \\
\quad \cup \{(tr' - tr, ref' \cup \{\checkmark\}) \mid (A \Box B)^n \land wait'\} & \\
\quad \cup \{((tr' - tr) \frown \langle \checkmark \rangle, ref') \mid \neg\, wait' \land (A \Box B)^n\} & \\
\quad \cup \{((tr' - tr) \frown \langle \checkmark \rangle, ref' \cup \{\checkmark\}) \mid \neg\, wait' \land (A \Box B)^n\} & \\
= \left\{ \begin{array}{l} (tr' - tr, ref') \\ \mid okay \land \neg\, wait \land okay' \land A \Box B \end{array} \right\} & [\text{PC}] \\
\quad \cup \left\{ \begin{array}{l} (tr' - tr, ref' \cup \{\checkmark\}) \\ \mid okay \land \neg\, wait \land okay' \land wait' \land A \Box B \end{array} \right\} & \\
\quad \cup \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref') \\ \mid okay \land \neg\, wait \land okay' \land \neg\, wait' \land A \Box B \end{array} \right\} & \\
\quad \cup \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref' \cup \{\checkmark\}) \\ \mid okay \land \neg\, wait \land okay' \land \neg\, wait' \land A \Box B \end{array} \right\} & \\
= \left\{ \begin{array}{l} (tr' - tr, ref') \\ \mid okay \land (A \Box B)^t_f \end{array} \right\} & [\text{Lemma J.17}] \\
\quad \cup \left\{ \begin{array}{l} (tr' - tr, ref' \cup \{\checkmark\}) \\ \mid okay \land wait' \land (A \Box B)^t_f \end{array} \right\} & \\
\quad \cup \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref') \\ \mid okay \land \neg\, wait' \land (A \Box B)^t_f \end{array} \right\} & \\
\quad \cup \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref' \cup \{\checkmark\}) \\ \mid okay \land \neg\, wait' \land (A \Box B)^t_f \end{array} \right\} &
\end{array}$$

$$
= \left\{ \begin{array}{l} (tr' - tr, ref') \\ \mid okay \wedge \mathbf{CSP1} \left(\begin{array}{l} (\neg\ (A)^f_f \wedge \neg\ (B)^f_f) \\ \Rightarrow \\ \left(\begin{array}{l} ((A)^t_f \wedge (B)^t_f) \\ \lhd tr' = tr \wedge wait' \rhd \\ ((A)^t_f \vee (B)^t_f) \end{array} \right) \end{array} \right) \end{array} \right\} \quad \text{[Lemma J.4]}
$$

$$
\cup \left\{ \begin{array}{l} (tr' - tr, ref' \cup \{\checkmark\}) \\ \mid okay \wedge wait' \wedge \mathbf{CSP1} \left(\begin{array}{l} (\neg\ (A)^f_f \wedge \neg\ (B)^f_f) \\ \Rightarrow \\ \left(\begin{array}{l} ((A)^t_f \wedge (B)^t_f) \\ \lhd tr' = tr \wedge wait' \rhd \\ ((A)^t_f \vee (B)^t_f) \end{array} \right) \end{array} \right) \end{array} \right\}
$$

$$
\cup \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref') \\ \mid okay \wedge \neg\ wait' \wedge \mathbf{CSP1} \left(\begin{array}{l} (\neg\ (A)^f_f \wedge \neg\ (B)^f_f) \\ \Rightarrow \\ \left(\begin{array}{l} ((A)^t_f \wedge (B)^t_f) \\ \lhd tr' = tr \wedge wait' \rhd \\ ((A)^t_f \vee (B)^t_f) \end{array} \right) \end{array} \right) \end{array} \right\}
$$

$$
\cup \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref' \cup \{\checkmark\}) \\ \mid okay \wedge \neg\ wait' \wedge \mathbf{CSP1} \left(\begin{array}{l} (\neg\ (A)^f_f \wedge \neg\ (B)^f_f) \\ \Rightarrow \\ \left(\begin{array}{l} ((A)^t_f \wedge (B)^t_f) \\ \lhd tr' = tr \wedge wait' \rhd \\ ((A)^t_f \vee (B)^t_f) \end{array} \right) \end{array} \right) \end{array} \right\}
$$

$$= \left\{ \begin{array}{l} (tr' - tr, ref') \\ \mid okay \land \left(\begin{array}{l} (\neg\, (A)^f_f \land \neg\, (B)^f_f) \\ \Rightarrow \\ \left(\begin{array}{l} ((A)^t_f \land (B)^t_f) \\ \lhd tr' = tr \land wait' \rhd \\ ((A)^t_f \lor (B)^t_f) \end{array} \right) \end{array} \right) \end{array} \right\} \quad \text{[Proviso 2 and PC]}$$

$$\cup \left\{ \begin{array}{l} (tr' - tr, ref' \cup \{\checkmark\}) \\ \mid okay \land wait' \land \left(\begin{array}{l} (\neg\, (A)^f_f \land \neg\, (B)^f_f) \\ \Rightarrow \\ \left(\begin{array}{l} ((A)^t_f \land (B)^t_f) \\ \lhd tr' = tr \land wait' \rhd \\ ((A)^t_f \lor (B)^t_f) \end{array} \right) \end{array} \right) \end{array} \right\}$$

$$\cup \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref') \\ \mid okay \land \neg\, wait' \land \left(\begin{array}{l} (\neg\, (A)^f_f \land \neg\, (B)^f_f) \\ \Rightarrow \\ \left(\begin{array}{l} ((A)^t_f \land (B)^t_f) \\ \lhd tr' = tr \land wait' \rhd \\ ((A)^t_f \lor (B)^t_f) \end{array} \right) \end{array} \right) \end{array} \right\}$$

$$\cup \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref' \cup \{\checkmark\}) \\ \mid okay \land \neg\, wait' \land \left(\begin{array}{l} (\neg\, (A)^f_f \land \neg\, (B)^f_f) \\ \Rightarrow \\ \left(\begin{array}{l} ((A)^t_f \land (B)^t_f) \\ \lhd tr' = tr \land wait' \rhd \\ ((A)^t_f \lor (B)^t_f) \end{array} \right) \end{array} \right) \end{array} \right\}$$

$$= \left\{ \begin{array}{l} (tr' - tr, ref') \\ \mid okay \land \left(\begin{array}{l} ((A)^t_f \land (B)^t_f) \\ \lhd tr' = tr \land wait' \rhd \\ ((A)^t_f \lor (B)^t_f) \end{array} \right) \end{array} \right\} \quad \text{[PC and SC]}$$

$$\cup \left\{ \begin{array}{l} (tr' - tr, ref' \cup \{\checkmark\}) \\ \mid okay \land wait' \land \left(\begin{array}{l} ((A)^t_f \land (B)^t_f) \\ \lhd tr' = tr \land wait' \rhd \\ ((A)^t_f \lor (B)^t_f) \end{array} \right) \end{array} \right\}$$

$$\cup \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref') \\ \mid okay \land \neg\, wait' \land \left(\begin{array}{l} ((A)^t_f \land (B)^t_f) \\ \lhd tr' = tr \land wait' \rhd \\ ((A)^t_f \lor (B)^t_f) \end{array} \right) \end{array} \right\}$$

$$\cup \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref' \cup \{\checkmark\}) \\ \mid okay \land \neg\, wait' \land \left(\begin{array}{l} ((A)^t_f \land (B)^t_f) \\ \lhd tr' = tr \land wait' \rhd \\ ((A)^t_f \lor (B)^t_f) \end{array} \right) \end{array} \right\}$$

$$
\begin{aligned}
&= \{(tr' - tr, ref') \mid okay \wedge tr' = tr \wedge wait' \wedge (A)^t_f \wedge (B)^t_f\} && [-]\\
&\quad \cup \{(tr' - tr, ref') \mid okay \wedge \neg\,(tr' = tr \wedge wait') \wedge (A)^t_f\}\\
&\quad \cup \{(tr' - tr, ref') \mid okay \wedge \neg\,(tr' = tr \wedge wait') \wedge (B)^t_f\}\\
&\quad \cup \{(tr' - tr, ref' \cup \{\checkmark\}) \mid okay \wedge wait' \wedge tr' = tr \wedge (A)^t_f \wedge (B)^t_f\}\\
&\quad \cup \{(tr' - tr, ref' \cup \{\checkmark\}) \mid okay \wedge wait' \wedge \neg\, tr' = tr \wedge (A)^t_f\}\\
&\quad \cup \{(tr' - tr, ref' \cup \{\checkmark\}) \mid okay \wedge wait' \wedge \neg\, tr' = tr \wedge (B)^t_f\}\\
&\quad \cup \{((tr' - tr) \frown \langle\checkmark\rangle, ref') \mid okay \wedge \neg\, wait' \wedge (A)^t_f\}\\
&\quad \cup \{((tr' - tr) \frown \langle\checkmark\rangle, ref') \mid okay \wedge \neg\, wait' \wedge (B)^t_f\}\\
&\quad \cup \{((tr' - tr) \frown \langle\checkmark\rangle, ref' \cup \{\checkmark\}) \mid okay \wedge \neg\, wait' \wedge (A)^t_f\}\\
&\quad \cup \{((tr' - tr) \frown \langle\checkmark\rangle, ref' \cup \{\checkmark\}) \mid okay \wedge \neg\, wait' \wedge (B)^t_f\}\\
&= \{(\langle\rangle, ref') \mid okay \wedge tr' = tr \wedge wait' \wedge (A)^t_f \wedge (B)^t_f\} && [\mathrm{ST}]\\
&\quad \cup \{(tr' - tr, ref') \mid okay \wedge \neg\,(tr' = tr \wedge wait') \wedge (A)^t_f\}\\
&\quad \cup \{(tr' - tr, ref') \mid okay \wedge \neg\,(tr' = tr \wedge wait') \wedge (B)^t_f\}\\
&\quad \cup \{(\langle\rangle, ref' \cup \{\checkmark\}) \mid okay \wedge wait' \wedge tr' = tr \wedge (A)^t_f \wedge (B)^t_f\}\\
&\quad \cup \{(tr' - tr, ref' \cup \{\checkmark\}) \mid okay \wedge wait' \wedge \neg\, tr' = tr \wedge (A)^t_f\}\\
&\quad \cup \{(tr' - tr, ref' \cup \{\checkmark\}) \mid okay \wedge wait' \wedge \neg\, tr' = tr \wedge (B)^t_f\}\\
&\quad \cup \{((tr' - tr) \frown \langle\checkmark\rangle, ref') \mid okay \wedge \neg\, wait' \wedge (A)^t_f\}\\
&\quad \cup \{((tr' - tr) \frown \langle\checkmark\rangle, ref') \mid okay \wedge \neg\, wait' \wedge (B)^t_f\}\\
&\quad \cup \{((tr' - tr) \frown \langle\checkmark\rangle, ref' \cup \{\checkmark\}) \mid okay \wedge \neg\, wait' \wedge (A)^t_f\}\\
&\quad \cup \{((tr' - tr) \frown \langle\checkmark\rangle, ref' \cup \{\checkmark\}) \mid okay \wedge \neg\, wait' \wedge (B)^t_f\}
\end{aligned}
$$

$$
\begin{aligned}
&= \{(\langle\rangle, ref') \mid okay \wedge tr' = tr \wedge wait' \wedge (A)^t_f \wedge (B)^t_f\} \\
&\quad \cup \{(\langle\rangle, ref' \cup \{\checkmark\}) \mid okay \wedge wait' \wedge tr' = tr \wedge (A)^t_f \wedge (B)^t_f\} \\
&\quad \cup \{(tr' - tr, ref') \mid okay \wedge \neg\,(tr' = tr \wedge wait') \wedge (A)^t_f\} \\
&\quad \cup \{(tr' - tr, ref') \mid okay \wedge \neg\,(tr' = tr \wedge wait') \wedge (B)^t_f\} \\
&\quad \cup \{(tr' - tr, ref' \cup \{\checkmark\}) \mid okay \wedge wait' \wedge \neg\, tr' = tr \wedge (A)^t_f\} \\
&\quad \cup \{(tr' - tr, ref' \cup \{\checkmark\}) \mid okay \wedge wait' \wedge \neg\, tr' = tr \wedge (B)^t_f\} \\
&\quad \cup \{((tr' - tr) \frown \langle\checkmark\rangle, ref') \mid okay \wedge \neg\, wait' \wedge (A)^t_f\} \\
&\quad \cup \{((tr' - tr) \frown \langle\checkmark\rangle, ref') \mid okay \wedge \neg\, wait' \wedge (B)^t_f\} \\
&\quad \cup \{((tr' - tr) \frown \langle\checkmark\rangle, ref' \cup \{\checkmark\}) \mid okay \wedge \neg\, wait' \wedge (A)^t_f\} \\
&\quad \cup \{((tr' - tr) \frown \langle\checkmark\rangle, ref' \cup \{\checkmark\}) \mid okay \wedge \neg\, wait' \wedge (B)^t_f\}
\end{aligned}
$$

[Lemma J.1]

$$
\begin{aligned}
&= \{(\langle\rangle, ref') \mid tr' = tr \wedge okay \wedge (A)^t_f \wedge (B)^t_f\} \qquad \text{[ST]} \\
&\quad \cup \{(\langle\rangle, ref') \mid tr' = tr \wedge okay \wedge \neg\, wait' \wedge (A)^t_f\}\} \\
&\quad \cup \{(\langle\rangle, ref') \mid tr' = tr \wedge okay \wedge \neg\, wait' \wedge (B)^t_f\}\} \\
&\quad \cup \{(tr' - tr, ref') \mid \neg\, tr' = tr \wedge okay \wedge (A)^t_f\} \\
&\quad \cup \{(tr' - tr, ref') \mid \neg\, tr' = tr \wedge okay \wedge (B)^t_f\} \\
&\quad \cup \{(\langle\rangle, ref' \cup \{\checkmark\}) \mid tr' = tr \wedge okay \wedge wait' \wedge (A)^t_f \wedge (B)^t_f\} \\
&\quad \cup \{(tr' - tr, ref' \cup \{\checkmark\}) \mid \neg\, tr' = tr \wedge okay \wedge wait' \wedge (A)^t_f\} \\
&\quad \cup \{(tr' - tr, ref' \cup \{\checkmark\}) \mid \neg\, tr' = tr \wedge okay \wedge wait' \wedge (B)^t_f\} \\
&\quad \cup \{((tr' - tr) \frown \langle\checkmark\rangle, ref') \mid okay \wedge \neg\, wait' \wedge (A)^t_f\} \\
&\quad \cup \{((tr' - tr) \frown \langle\checkmark\rangle, ref') \mid okay \wedge \neg\, wait' \wedge (B)^t_f\} \\
&\quad \cup \{((tr' - tr) \frown \langle\checkmark\rangle, ref' \cup \{\checkmark\}) \mid okay \wedge \neg\, wait' \wedge (A)^t_f\} \\
&\quad \cup \{((tr' - tr) \frown \langle\checkmark\rangle, ref' \cup \{\checkmark\}) \mid okay \wedge \neg\, wait' \wedge (B)^t_f\}
\end{aligned}
$$

$$
\begin{aligned}
&= \{(\langle\rangle, ref') \mid tr' = tr \wedge okay \wedge (A)^t_f \wedge (B)^t_f\} \\
&\quad \cup \{(\langle\rangle, ref' \cup \{\checkmark\}) \mid tr' = tr \wedge okay \wedge wait' \wedge (A)^t_f \wedge (B)^t_f\} \\
&\quad \cup \{(tr' - tr, ref') \mid \neg\, tr' = tr \wedge okay \wedge (A)^t_f\} \\
&\quad \cup \{(tr' - tr, ref' \cup \{\checkmark\}) \mid \neg\, tr' = tr \wedge okay \wedge wait' \wedge (A)^t_f\} \\
&\quad \cup \{((tr' - tr) \frown \langle\checkmark\rangle, ref') \mid okay \wedge \neg\, wait' \wedge (A)^t_f\} \\
&\quad \cup \{((tr' - tr) \frown \langle\checkmark\rangle, ref' \cup \{\checkmark\}) \mid okay \wedge \neg\, wait' \wedge (A)^t_f\} \\
&\quad \cup \{(tr' - tr, ref') \mid \neg\, tr' = tr \wedge okay \wedge (B)^t_f\} \\
&\quad \cup \{(tr' - tr, ref' \cup \{\checkmark\}) \mid \neg\, tr' = tr \wedge okay \wedge wait' \wedge (B)^t_f\} \\
&\quad \cup \{((tr' - tr) \frown \langle\checkmark\rangle, ref') \mid okay \wedge \neg\, wait' \wedge (B)^t_f\} \\
&\quad \cup \{((tr' - tr) \frown \langle\checkmark\rangle, ref' \cup \{\checkmark\}) \mid okay \wedge \neg\, wait' \wedge (B)^t_f\} \\
&\quad \cup \{(\langle\rangle, ref') \mid tr' = tr \wedge okay \wedge \neg\, wait' \wedge (A)^t_f\}\} \\
&\quad \cup \{(\langle\rangle, ref') \mid tr' = tr \wedge okay \wedge \neg\, wait' \wedge (B)^t_f\}\}
\end{aligned}
$$

[Lemmas J.22, J.23, and J.24]

$$\begin{aligned}
&= \{(\langle\rangle, X) \mid (\langle\rangle, X) \in \mathit{failures}(\Upsilon(A)) \cap \mathit{failures}(\Upsilon(B))\} && [\mathit{failures}]\\
&\quad \cup\{(s, X) \mid (s, X) \in \mathit{failures}(\Upsilon(A)) \cup \mathit{failures}(\Upsilon(B)) \land s \neq \langle\rangle\}\\
&\quad \cup\{(\langle\rangle, X) \mid X \subseteq \Sigma \land \langle\checkmark\rangle \in \mathit{traces}(\Upsilon(A)) \cup \mathit{traces}(\Upsilon(B))\}\\
&= \mathit{failures}(\Upsilon(A) \Box \Upsilon(B)) && [\Upsilon]\\
&= \mathit{failures}(A \Box B)
\end{aligned}$$

Theorem J.30

$$\mathit{failures}^{\mathcal{UTP}}(\Box\, x : S \bullet A) = \mathit{failures}(\Upsilon(\Box\, x : S \bullet A))$$

provided

1. $\forall\, i : S \bullet A[v_i/x]$ *is* $\boldsymbol{R}$
2. $\forall\, i : S \bullet A[v_i/x]$ *is divergence-free*

Inductive Hypothesis (A):

$$\forall\, i : S \bullet \mathit{failures}^{\mathcal{UTP}}(A[v_i/x]) = \mathit{failures}(\Upsilon(A)[v_i/x])$$

Proof. By induction on S

Base Case. $S = \{\}$
Proof.

$$\begin{aligned}
&\mathit{failures}^{\mathcal{UTP}}(\Box\, x : S \bullet A) && [\text{Assumption}]\\
&= \mathit{failures}^{\mathcal{UTP}}(\Box\, x : \{\} \bullet A) && [\text{Property of } \Box]\\
&= \mathit{failures}^{\mathcal{UTP}}(\mathit{Stop}) && [\text{Theorem J.22}]\\
&= \mathit{failures}(\Upsilon(\mathit{Stop})) && [\text{Property of } \Box]\\
&= \mathit{failures}(\Upsilon(\Box\, x : \{\} \bullet A)) && [\text{Assumption}]\\
&= \mathit{failures}(\Upsilon(\Box\, x : S \bullet A))
\end{aligned}$$

Inductive Hypothesis (S):

$$\mathit{failures}^{\mathcal{UTP}}(\Box\, x : S \bullet A) = \mathit{failures}(\Upsilon(\Box\, x : S \bullet A))$$

Inductive Step

$$\mathit{failures}^{\mathcal{UTP}}(\Box\, x : S \cup \{v_i\} \bullet A) = \mathit{failures}(\Upsilon(\Box\, x : S \cup \{v_i\} \bullet A))$$

Proof.

$$\begin{aligned}
&\mathit{failures}^{\mathcal{UTP}}(\Box\, x : S \cup \{v_i\} \bullet A) && [\Box]\\
&= \mathit{failures}^{\mathcal{UTP}}(A[v_i/x] \,\Box\, (\Box\, x : S \setminus \{v_i\} \bullet A))\\
& && [\text{Theorem J.29 (Provisos, IH-}A\text{ and IH-}S)]\\
&= \mathit{failures}(\Upsilon(A[v_i/x] \,\Box\, (\Box\, x : S \setminus \{v_i\} \bullet A))) && [\Box]\\
&= \mathit{failures}(\Upsilon(\Box\, x : S \cup \{v_i\} \bullet A))
\end{aligned}$$

Theorem J.31 $\mathit{failures}^{\mathcal{UTP}}(A \sqcap B) = \mathit{failures}(\Upsilon(A \sqcap B))$

Inductive Hypothesis:

$$\begin{aligned}
&\mathit{failures}^{\mathcal{UTP}}(A) = \mathit{failures}(\Upsilon(A))\\
&\mathit{failures}^{\mathcal{UTP}}(B) = \mathit{failures}(\Upsilon(B))
\end{aligned}$$

Proof.

$$\begin{aligned}
&\mathit{failures}^{\mathcal{UTP}}(A \sqcap B) && [\mathit{failures}^{\mathcal{UTP}}]\\
&= \{(\mathit{tr}' - \mathit{tr}, \mathit{ref}') \mid (A \sqcap B)^n\} && [A^t]\\
&\quad \cup \{(\mathit{tr}' - \mathit{tr}, \mathit{ref}' \cup \{\checkmark\}) \mid (A \sqcap B)^n \wedge \mathit{wait}'\}\\
&\quad \cup \{((\mathit{tr}' - \mathit{tr}) \frown \langle\checkmark\rangle, \mathit{ref}') \mid (A \sqcap B)^t\}\\
&\quad \cup \{((\mathit{tr}' - \mathit{tr}) \frown \langle\checkmark\rangle, \mathit{ref}' \cup \{\checkmark\}) \mid (A \sqcap B)^t\}\\
&= \{(\mathit{tr}' - \mathit{tr}, \mathit{ref}') \mid (A \sqcap B)^n\} && [A^n]\\
&\quad \cup \{(\mathit{tr}' - \mathit{tr}, \mathit{ref}' \cup \{\checkmark\}) \mid (A \sqcap B)^n \wedge \mathit{wait}'\}\\
&\quad \cup \{((\mathit{tr}' - \mathit{tr}) \frown \langle\checkmark\rangle, \mathit{ref}') \mid \neg\, \mathit{wait}' \wedge (A \sqcap B)^n\}\\
&\quad \cup \{((\mathit{tr}' - \mathit{tr}) \frown \langle\checkmark\rangle, \mathit{ref}' \cup \{\checkmark\}) \mid \neg\, \mathit{wait}' \wedge (A \sqcap B)^n\}\\
&= \left\{\begin{array}{l} (\mathit{tr}' - \mathit{tr}, \mathit{ref}')\\ \mid \mathit{okay} \wedge \neg\, \mathit{wait} \wedge \mathit{okay}' \wedge A \sqcap B \end{array}\right\} && [\text{PC}]\\
&\quad \cup \left\{\begin{array}{l} (\mathit{tr}' - \mathit{tr}, \mathit{ref}' \cup \{\checkmark\})\\ \mid \mathit{okay} \wedge \neg\, \mathit{wait} \wedge \mathit{okay}' \wedge \mathit{wait}' \wedge A \sqcap B \end{array}\right\}\\
&\quad \cup \left\{\begin{array}{l} ((\mathit{tr}' - \mathit{tr}) \frown \langle\checkmark\rangle, \mathit{ref}')\\ \mid \mathit{okay} \wedge \neg\, \mathit{wait} \wedge \mathit{okay}' \wedge \neg\, \mathit{wait}' \wedge A \sqcap B \end{array}\right\}\\
&\quad \cup \left\{\begin{array}{l} ((\mathit{tr}' - \mathit{tr}) \frown \langle\checkmark\rangle, \mathit{ref}' \cup \{\checkmark\})\\ \mid \mathit{okay} \wedge \neg\, \mathit{wait} \wedge \mathit{okay}' \wedge \neg\, \mathit{wait}' \wedge A \sqcap B \end{array}\right\}
\end{aligned}$$

$$
\begin{aligned}
&= \left\{ \begin{array}{l} (tr' - tr, ref') \\ \mid okay \wedge (A \sqcap B)^t_f \end{array} \right\} && [\,] \\
&\quad \cup \left\{ \begin{array}{l} (tr' - tr, ref' \cup \{\checkmark\}) \\ \mid okay \wedge wait' \wedge (A \sqcap B)^t_f \end{array} \right\} \\
&\quad \cup \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref') \\ \mid okay \wedge \neg\, wait' \wedge (A \sqcap B)^t_f \end{array} \right\} \\
&\quad \cup \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref' \cup \{\checkmark\}) \\ \mid okay \wedge \neg\, wait' \wedge (A \sqcap B)^t_f \end{array} \right\} \\
&= \left\{ \begin{array}{l} (tr' - tr, ref') \\ \mid okay \wedge (A \vee B)^t_f \end{array} \right\} && [\mathrm{PC}] \\
&\quad \cup \left\{ \begin{array}{l} (tr' - tr, ref' \cup \{\checkmark\}) \\ \mid okay \wedge wait' \wedge (A \vee B)^t_f \end{array} \right\} \\
&\quad \cup \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref') \\ \mid okay \wedge \neg\, wait' \wedge (A \vee B)^t_f \end{array} \right\} \\
&\quad \cup \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref' \cup \{\checkmark\}) \\ \mid okay \wedge \neg\, wait' \wedge (A \vee B)^t_f \end{array} \right\} \\
&= \left\{ \begin{array}{l} (tr' - tr, ref') \\ \mid okay \wedge (A)^t_f \vee (B)^t_f \end{array} \right\} && [\mathrm{PC, SC\ and\ ST}] \\
&\quad \cup \left\{ \begin{array}{l} (tr' - tr, ref' \cup \{\checkmark\}) \\ \mid okay \wedge wait' \wedge (A)^t_f \vee (B)^t_f \end{array} \right\} \\
&\quad \cup \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref') \\ \mid okay \wedge \neg\, wait' \wedge (A)^t_f \vee (B)^t_f \end{array} \right\} \\
&\quad \cup \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref' \cup \{\checkmark\}) \\ \mid okay \wedge \neg\, wait' \wedge (A)^t_f \vee (B)^t_f \end{array} \right\} \\
&= \{(tr' - tr, ref') \mid okay \wedge (A)^t_f\} && [failures^{\mathcal{UTP}}] \\
&\quad \cup \{(tr' - tr, ref' \cup \{\checkmark\}) \mid okay \wedge wait' \wedge (A)^t_f\} \\
&\quad \cup \{((tr' - tr) \frown \langle \checkmark \rangle, ref') \mid okay \wedge \neg\, wait' \wedge (A)^t_f\} \\
&\quad \cup \{((tr' - tr) \frown \langle \checkmark \rangle, ref' \cup \{\checkmark\}) \mid okay \wedge \neg\, wait' \wedge (A)^t_f\} \\
&\quad \cup \{(tr' - tr, ref') \mid okay \wedge (B)^t_f\} \\
&\quad \cup \{(tr' - tr, ref' \cup \{\checkmark\}) \mid okay \wedge wait' \wedge (B)^t_f\} \\
&\quad \cup \{((tr' - tr) \frown \langle \checkmark \rangle, ref') \mid okay \wedge \neg\, wait' \wedge (B)^t_f\} \\
&\quad \cup \{((tr' - tr) \frown \langle \checkmark \rangle, ref' \cup \{\checkmark\}) \mid okay \wedge \neg\, wait' \wedge (B)^t_f\} \\
&= failures^{\mathcal{UTP}}(\Upsilon(A)) \cup failures^{\mathcal{UTP}}(\Upsilon(A)) && [\mathrm{IH}] \\
&= failures(\Upsilon(A)) \cup failures(\Upsilon(B)) && [failures] \\
&= failures(\Upsilon(A) \sqcap \Upsilon(B)) && [\Upsilon] \\
&= failures(\Upsilon(A \sqcap B))
\end{aligned}
$$

Theorem J.32

$$\mathit{failures}^{\mathcal{UTP}}(\sqcap x : S \bullet A) = \mathit{failures}(\Upsilon(\sqcap x : S \bullet A))$$

provided

1. $\forall i : S \bullet A[v_i/x]$ *is* $\boldsymbol{R}$
2. $\forall i : S \bullet A[v_i/x]$ *is divergence-free*
3. $S \neq \{\}$

Inductive Hypothesis (A):

$$\forall i : S \bullet \mathit{failures}^{\mathcal{UTP}}(A[v_i/x]) = \mathit{failures}(\Upsilon(A)[v_i/x])$$

Proof. By induction on S

Base Case. $S = \{v\}$
Proof.

$$\begin{aligned}
&\mathit{failures}^{\mathcal{UTP}}(\sqcap x : S \bullet A) && [\text{Assumption}]\\
&= \mathit{failures}^{\mathcal{UTP}}(\sqcap x : \{v\} \bullet A) && [\sqcap]\\
&= \mathit{failures}^{\mathcal{UTP}}(A[v/x]) && [\text{IH}]\\
&= \mathit{failures}(\Upsilon(A[v/x])) && [\sqcap]\\
&= \mathit{failures}(\Upsilon(\sqcap x : \{v\} \bullet A)) && [\text{Assumption}]\\
&= \mathit{failures}(\Upsilon(\sqcap x : S \bullet A))
\end{aligned}$$

Inductive Hypothesis (S):

$$\mathit{failures}^{\mathcal{UTP}}(\sqcap x : S \bullet A) = \mathit{failures}(\Upsilon(\sqcap x : S \bullet A))$$

Inductive Step

$$\mathit{failures}^{\mathcal{UTP}}(\sqcap x : S \cup \{v_i\} \bullet A) = \mathit{failures}(\Upsilon(\sqcap x : S \cup \{v_i\} \bullet A))$$

Proof.

$$\begin{aligned}
&\mathit{failures}^{\mathcal{UTP}}(\sqcap x : S \cup \{v_i\} \bullet A) && [\sqcap]\\
&= \mathit{failures}^{\mathcal{UTP}}(A[v_i/x] \mathbin{\Box} (\sqcap x : S \setminus \{v_i\} \bullet A))
\end{aligned}$$

[Theorem J.31 (Provisos, IH-A and IH-S)]

$= \mathit{failures}(\Upsilon(A[v_i/x] \Box (\sqcap x : S \setminus \{v_i\} \bullet A)))$ [$\sqcap$]

$= \mathit{failures}(\Upsilon(\sqcap x : S \cup \{v_i\} \bullet A))$

Theorem J.33 $\mathit{failures}^{\mathcal{UTP}}(g \;\&\; A) = \mathit{failures}(\Upsilon(g \;\&\; A))$

Inductive Hypothesis:

$\mathit{failures}^{\mathcal{UTP}}(A) = \mathit{failures}(\Upsilon(A))$

Proof. The proof will be conducted by cases on g.

Case 1. g is *false*
Proof.

$\mathit{failures}^{\mathcal{UTP}}(g \;\&\; A)$ [Assumption]

$= \mathit{failures}^{\mathcal{UTP}}(\mathit{false} \;\&\; A)$ [Law 38]

$= \mathit{failures}^{\mathcal{UTP}}(\mathit{Stop})$ [Theorem J.22]

$= \mathit{failures}(\Upsilon(\mathit{Stop}))$ [Law 38]

$= \mathit{failures}(\Upsilon(\mathit{false} \;\&\; A))$ [Assumption]

$= \mathit{failures}(\Upsilon(g \;\&\; A))$

Case 2. g is *true*
Proof.

$\mathit{failures}^{\mathcal{UTP}}(g \;\&\; A)$ [Assumption]

$= \mathit{failures}^{\mathcal{UTP}}(\mathit{true} \;\&\; A)$ [Law 37]

$= \mathit{failures}^{\mathcal{UTP}}(A)$ [IH]

$= \mathit{failures}(\Upsilon(A))$ [Law 37]

$= \mathit{failures}(\Upsilon(\mathit{true} \;\&\; A))$ [Assumption]

$= \mathit{failures}(\Upsilon(g \;\&\; A))$

Theorem J.34 $\mathit{failures}^{\mathcal{UTP}}(P;\ Q) = \mathit{failures}(\Upsilon(P;\ Q))$
provided

1. *P and Q are divergence-free*

2. *$P = \boldsymbol{R}(P_{pre} \vdash P_{post})$ and $Q = \boldsymbol{R}(Q_{pre} \vdash Q_{post})$*
3. *P_{pre} does not mention any dashed variable*
4. *P_{post} and Q_{post} are **R1** and **R2***

Inductive Hypothesis:

$$\begin{array}{l} \mathit{failures}^{\mathcal{UTP}}(P) = \mathit{failures}(\Upsilon(P)) \\ \text{and} \\ \mathit{failures}^{\mathcal{UTP}}(Q) = \mathit{failures}(\Upsilon(Q)) \end{array}$$

Proof.

$$\begin{array}{ll}
\mathit{failures}^{\mathcal{UTP}}(P;\ Q) & [\mathit{failures}^{\mathcal{UTP}}] \\
= \{(tr' - tr, ref') \mid (P;\ Q)^n\} & [A^t] \\
\quad \cup \{(tr' - tr, ref' \cup \{\checkmark\}) \mid (P;\ Q)^n \land wait'\} & \\
\quad \cup \{((tr' - tr) ^\frown \langle \checkmark \rangle, ref') \mid (P;\ Q)^t\} & \\
\quad \cup \{((tr' - tr) ^\frown \langle \checkmark \rangle, ref' \cup \{\checkmark\}) \mid (P;\ Q)^t\} & \\
= \{(tr' - tr, ref') \mid (P;\ Q)^n\} & [A^n] \\
\quad \cup \{(tr' - tr, ref' \cup \{\checkmark\}) \mid (P;\ Q)^n \land wait'\} & \\
\quad \cup \{((tr' - tr) ^\frown \langle \checkmark \rangle, ref') \mid \neg\, wait' \land (P;\ Q)^n\} & \\
\quad \cup \{((tr' - tr) ^\frown \langle \checkmark \rangle, ref' \cup \{\checkmark\}) \mid \neg\, wait' \land (P;\ Q)^n\} & \\
= \{(tr' - tr, ref') \mid okay \land \neg\, wait \land okay' \land (P;\ Q)\} & [\text{PC}] \\
\quad \cup \{(tr' - tr, ref' \cup \{\checkmark\}) \mid okay \land \neg\, wait \land okay' \land (P;\ Q) \land wait'\} & \\
\quad \cup \{((tr' - tr) ^\frown \langle \checkmark \rangle, ref') \mid \neg\, wait' \land okay \land \neg\, wait \land okay' \land (P;\ Q)\} & \\
\quad \cup \{((tr' - tr) ^\frown \langle \checkmark \rangle, ref' \cup \{\checkmark\}) \mid \neg\, wait' \land okay \land \neg\, wait \land okay' \land (P;\ Q)\} & \\
= \{(tr' - tr, ref') \mid okay \land (P;\ Q)^t_f\} & [\text{Lemma J.19 (Assumptions)}] \\
\quad \cup \{(tr' - tr, ref' \cup \{\checkmark\}) \mid okay \land wait' \land (P;\ Q)^t_f\} & \\
\quad \cup \{((tr' - tr) ^\frown \langle \checkmark \rangle, ref') \mid \neg\, wait' \land okay \land (P;\ Q)^t_f\} & \\
\quad \cup \{((tr' - tr) ^\frown \langle \checkmark \rangle, ref' \cup \{\checkmark\}) \mid \neg\, wait' \land okay \land (P;\ Q)^t_f\} &
\end{array}$$

$$
\begin{aligned}
&= \left\{ \begin{array}{l} (tr' - tr, ref') \mid \\ \qquad okay \wedge \mathbf{CSP1}\left(\begin{array}{l}(wait' \wedge P_{post}) \\ \vee ((okay' \wedge \neg\, wait' \wedge P_{post});\ Q_{post})\end{array}\right) \end{array} \right\} \\
&\quad \cup \left\{ \begin{array}{l} (tr' - tr, ref' \cup \{\checkmark\}) \mid \\ \qquad okay \wedge wait' \wedge \mathbf{CSP1}\left(\begin{array}{l}(wait' \wedge P_{post}) \\ \vee ((okay' \wedge \neg\, wait' \wedge P_{post});\ Q_{post})\end{array}\right) \end{array} \right\} \\
&\quad \cup \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref') \mid \\ \qquad okay \wedge \neg\, wait' \wedge \mathbf{CSP1}\left(\begin{array}{l}(wait' \wedge P_{post}) \\ \vee ((okay' \wedge \neg\, wait' \wedge P_{post});\ Q_{post})\end{array}\right) \end{array} \right\} \\
&\quad \cup \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref' \cup \{\checkmark\}) \mid \\ \qquad okay \wedge \neg\, wait' \wedge \mathbf{CSP1}\left(\begin{array}{l}(wait' \wedge P_{post}) \\ \vee ((okay' \wedge \neg\, wait' \wedge P_{post});\ Q_{post})\end{array}\right) \end{array} \right\}
\end{aligned}
$$

[Lemma J.4, PC, SC and ST]

$$
\begin{aligned}
&= \{(tr' - tr, ref') \mid okay \wedge wait' \wedge P_{post}\} \\
&\quad \cup \{(tr' - tr, ref') \mid okay \wedge ((okay' \wedge \neg\, wait' \wedge P_{post});\ Q_{post})\} \\
&\quad \cup \{(tr' - tr, ref' \cup \{\checkmark\}) \mid okay \wedge wait' \wedge P_{post}\} \\
&\quad \cup \{(tr' - tr, ref' \cup \{\checkmark\}) \mid okay \wedge wait' \wedge ((okay' \wedge \neg\, wait' \wedge P_{post});\ Q_{post})\} \\
&\quad \cup \{((tr' - tr) \frown \langle \checkmark \rangle, ref') \mid okay \wedge \neg\, wait' \wedge ((okay' \wedge \neg\, wait' \wedge P_{post});\ Q_{post})\} \\
&\quad \cup \{((tr' - tr) \frown \langle \checkmark \rangle, ref' \cup \{\checkmark\}) \mid okay \wedge \neg\, wait' \wedge ((okay' \wedge \neg\, wait' \wedge P_{post});\ Q_{post})\}
\end{aligned}
$$

[Sequence and PC]

$$
\begin{aligned}
&= \{(tr' - tr, ref') \mid okay \wedge wait' \wedge P_{post}\} \\
&\quad \cup \{(tr' - tr, ref') \mid (okay \wedge \neg\, wait' \wedge P_{post});\ (okay \wedge Q_{post})\} \\
&\quad \cup \{(tr' - tr, ref' \cup \{\checkmark\}) \mid okay \wedge wait' \wedge P_{post}\} \\
&\quad \cup \{(tr' - tr, ref' \cup \{\checkmark\}) \mid (okay \wedge \neg\, wait' \wedge P_{post});\ (okay \wedge wait' \wedge Q_{post})\} \\
&\quad \cup \{((tr' - tr) \frown \langle \checkmark \rangle, ref') \mid (okay \wedge \neg\, wait' \wedge P_{post});\ (okay \wedge \neg\, wait' \wedge Q_{post})\} \\
&\quad \cup \{((tr' - tr) \frown \langle \checkmark \rangle, ref' \cup \{\checkmark\}) \mid (okay \wedge \neg\, wait' \wedge P_{post});\ (okay \wedge \neg\, wait' \wedge Q_{post})\}
\end{aligned}
$$

[Sequence, PC and SC]

$$
\begin{aligned}
&= \{(tr' - tr, ref') \mid okay \wedge wait' \wedge P_{post}\} \\
&\quad \cup \{(tr' - tr, ref' \cup \{\checkmark\}) \mid okay \wedge wait' \wedge P_{post}\} \\
&\quad \cup \left\{ \begin{array}{l} (s \frown t, X) \mid \\ \qquad s \in \{tr' - tr \mid okay \wedge \neg\, wait' \wedge P_{post}\} \\ \qquad \wedge\, (t, X) \in \{(tr' - tr, ref') \mid okay \wedge Q_{post}\} \end{array} \right\} \\
&\quad \cup \left\{ \begin{array}{l} (s \frown t, X \cup \{\checkmark\}) \mid \\ \qquad s \in \{tr' - tr \mid okay \wedge \neg\, wait' \wedge P_{post}\} \\ \qquad \wedge\, (t, X) \in \{(tr' - tr, ref') \mid okay \wedge wait' \wedge Q_{post}\} \end{array} \right\} \\
&\quad \cup \left\{ \begin{array}{l} (s \frown t \frown \langle \checkmark \rangle, X) \mid \\ \qquad s \in \{tr' - tr \mid okay \wedge \neg\, wait' \wedge P_{post}\} \\ \qquad \wedge\, (t, X) \in \{(tr' - tr, ref') \mid okay \wedge \neg\, wait' \wedge Q_{post}\} \end{array} \right\} \\
&\quad \cup \left\{ \begin{array}{l} (s \frown t \frown \langle \checkmark \rangle, X \cup \{\checkmark\}) \mid \\ \qquad s \in \{tr' - tr \mid okay \wedge \neg\, wait' \wedge P_{post}\} \\ \qquad \wedge\, (t, X) \in \{(tr' - tr, ref') \mid okay \wedge \neg\, wait' \wedge Q_{post}\} \end{array} \right\}
\end{aligned}
$$

[SC]

$$
\begin{aligned}
&= \{(tr' - tr, ref') \mid okay \wedge wait' \wedge P_{post}\} \\
&\quad \cup \{(tr' - tr, ref' \cup \{\checkmark\}) \mid okay \wedge wait' \wedge P_{post}\} \\
&\quad \cup \left\{ \begin{array}{l} (s \frown t, X) \mid \\ \qquad s \in \{(tr' - tr) \mid okay \wedge \neg\, wait' \wedge P_{post}\} \\ \qquad \wedge\, (t, X) \in \{(tr' - tr, ref') \mid okay \wedge Q_{post}\} \end{array} \right\} \\
&\quad \cup \left\{ \begin{array}{l} (s \frown t, X) \mid \\ \qquad s \in \{(tr' - tr) \mid okay \wedge \neg\, wait' \wedge P_{post}\} \\ \qquad \wedge\, (t, X) \in \{(tr' - tr, ref' \cup \{\checkmark\}) \mid okay \wedge wait' \wedge Q_{post}\} \end{array} \right\} \\
&\quad \cup \left\{ \begin{array}{l} (s \frown t, X) \mid \\ \qquad s \in \{(tr' - tr) \mid okay \wedge \neg\, wait' \wedge P_{post}\} \\ \qquad \wedge\, (t, X) \in \{((tr' - tr) \frown \langle \checkmark \rangle, ref') \mid okay \wedge \neg\, wait' \wedge Q_{post}\} \end{array} \right\} \\
&\quad \cup \left\{ \begin{array}{l} (s \frown t, X) \mid \\ \qquad s \in \{(tr' - tr) \mid okay \wedge \neg\, wait' \wedge P_{post}\} \\ \qquad \wedge\, (t, X) \in \{((tr' - tr) \frown \langle \checkmark \rangle, ref' \cup \{\checkmark\}) \mid okay \wedge \neg\, wait' \wedge Q_{post}\} \end{array} \right\}
\end{aligned}
$$

[Lemma J.4 and Assumption 4]

$$
\begin{array}{l}
= \{(tr' - tr, ref') \mid okay \land wait' \land \mathbf{CSP1}(\mathbf{R1}(\mathbf{R2}(P_{post})))\} \\
\quad \cup \{(tr' - tr, ref' \cup \{\checkmark\}) \mid okay \land wait' \land \mathbf{CSP1}(\mathbf{R1}(\mathbf{R2}(P_{post})))\} \\
\quad \cup \left\{ \begin{array}{l} (s \frown t, X) \mid \\ \qquad s \in \{(tr' - tr) \mid okay \land \neg\, wait' \land \mathbf{CSP1}(\mathbf{R1}(\mathbf{R2}(P_{post})))\} \\ \qquad \land (t, X) \in \{(tr' - tr, ref') \mid okay \land \mathbf{CSP1}(\mathbf{R1}(\mathbf{R2}(Q_{post})))\} \end{array} \right\} \\
\quad \cup \left\{ \begin{array}{l} (s \frown t, X) \mid \\ \qquad s \in \{(tr' - tr) \mid okay \land \neg\, wait' \land \mathbf{CSP1}(\mathbf{R1}(\mathbf{R2}(P_{post})))\} \\ \qquad \land (t, X) \in \left\{ \begin{array}{l} (tr' - tr, ref' \cup \{\checkmark\}) \mid \\ \qquad okay \land wait' \land \mathbf{CSP1}(\mathbf{R1}(\mathbf{R2}(Q_{post}))) \end{array} \right\} \end{array} \right\} \\
\quad \cup \left\{ \begin{array}{l} (s \frown t, X) \mid \\ \qquad s \in \{(tr' - tr) \mid okay \land \neg\, wait' \land \mathbf{CSP1}(\mathbf{R1}(\mathbf{R2}(P_{post})))\} \\ \qquad \land (t, X) \in \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref') \mid \\ \qquad okay \land \neg\, wait' \land \mathbf{CSP1}(\mathbf{R1}(\mathbf{R2}(Q_{post}))) \end{array} \right\} \end{array} \right\} \\
\quad \cup \left\{ \begin{array}{l} (s \frown t, X) \mid \\ \qquad s \in \{(tr' - tr) \mid okay \land \neg\, wait' \land \mathbf{CSP1}(\mathbf{R1}(\mathbf{R2}(P_{post})))\} \\ \qquad \land (t, X) \in \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref' \cup \{\checkmark\}) \mid \\ \qquad okay \land \neg\, wait' \land \mathbf{CSP1}(\mathbf{R1}(\mathbf{R2}(Q_{post}))) \end{array} \right\} \end{array} \right\}
\end{array}
$$

[Lemma J.8 and PC]

$$
\begin{array}{l}
= \{(tr' - tr, ref') \mid okay \land wait' \land (\mathbf{R}(true \vdash P_{post}))^t_f\} \\
\quad \cup \{(tr' - tr, ref' \cup \{\checkmark\}) \mid okay \land wait' \land (\mathbf{R}(true \vdash P_{post}))^t_f\} \\
\quad \cup \left\{ \begin{array}{l} (s \frown t, X) \mid \\ \qquad s \in \{(tr' - tr) \mid okay \land \neg\, wait' \land (\mathbf{R}(true \vdash P_{post}))^t_f\} \\ \qquad \land (t, X) \in \{(tr' - tr, ref') \mid okay \land (\mathbf{R}(true \vdash Q_{post}))^t_f\} \end{array} \right\} \\
\quad \cup \left\{ \begin{array}{l} (s \frown t, X) \mid \\ \qquad s \in \{(tr' - tr) \mid okay \land \neg\, wait' \land (\mathbf{R}(true \vdash P_{post}))^t_f\} \\ \qquad \land (t, X) \in \left\{ \begin{array}{l} (tr' - tr, ref' \cup \{\checkmark\}) \mid \\ \qquad okay \land wait' \land (\mathbf{R}(true \vdash Q_{post}))^t_f \end{array} \right\} \end{array} \right\} \\
\quad \cup \left\{ \begin{array}{l} (s \frown t, X) \mid \\ \qquad s \in \{(tr' - tr) \mid okay \land \neg\, wait' \land (\mathbf{R}(true \vdash P_{post}))^t_f\} \\ \qquad \land (t, X) \in \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref') \mid \\ \qquad okay \land \neg\, wait' \land (\mathbf{R}(true \vdash Q_{post}))^t_f \end{array} \right\} \end{array} \right\} \\
\quad \cup \left\{ \begin{array}{l} (s \frown t, X) \mid \\ \qquad s \in \{(tr' - tr) \mid okay \land \neg\, wait' \land (\mathbf{R}(true \vdash P_{post}))^t_f\} \\ \qquad \land (t, X) \in \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref' \cup \{\checkmark\}) \mid \\ \qquad okay \land \neg\, wait' \land (\mathbf{R}(true \vdash Q_{post}))^t_f \end{array} \right\} \end{array} \right\}
\end{array}
$$

[Assumption 1]

$$
\begin{aligned}
&= \{(tr' - tr, ref') \mid okay \wedge wait' \wedge (\mathbf{R}(P_{pre} \vdash P_{post}))^t_f\} \\
&\cup \{(tr' - tr, ref' \cup \{\checkmark\}) \mid okay \wedge wait' \wedge (\mathbf{R}(P_{pre} \vdash P_{post}))^t_f\} \\
&\cup \left\{ \begin{array}{l} (s \frown t, X) \mid \\ \qquad s \in \{(tr' - tr) \mid okay \wedge \neg\, wait' \wedge (\mathbf{R}(P_{pre} \vdash P_{post}))^t_f\} \\ \qquad \wedge (t, X) \in \{(tr' - tr, ref') \mid okay \wedge (\mathbf{R}(Q_{pre} \vdash Q_{post}))^t_f\} \end{array} \right\} \\
&\cup \left\{ \begin{array}{l} (s \frown t, X) \mid \\ \qquad s \in \{(tr' - tr) \mid okay \wedge \neg\, wait' \wedge (\mathbf{R}(P_{pre} \vdash P_{post}))^t_f\} \\ \qquad \wedge (t, X) \in \left\{ \begin{array}{l} (tr' - tr, ref' \cup \{\checkmark\}) \mid \\ \qquad okay \wedge wait' \wedge (\mathbf{R}(Q_{pre} \vdash Q_{post}))^t_f \end{array} \right\} \end{array} \right\} \\
&\cup \left\{ \begin{array}{l} (s \frown t, X) \mid \\ \qquad s \in \{(tr' - tr) \mid okay \wedge \neg\, wait' \wedge (\mathbf{R}(P_{pre} \vdash P_{post}))^t_f\} \\ \qquad \wedge (t, X) \in \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref') \mid \\ \qquad okay \wedge \neg\, wait' \wedge (\mathbf{R}(Q_{pre} \vdash Q_{post}))^t_f \end{array} \right\} \end{array} \right\} \\
&\cup \left\{ \begin{array}{l} (s \frown t, X) \mid \\ \qquad s \in \{(tr' - tr) \mid okay \wedge \neg\, wait' \wedge (\mathbf{R}(P_{pre} \vdash P_{post}))^t_f\} \\ \qquad \wedge (t, X) \in \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref' \cup \{\checkmark\}) \mid \\ \qquad okay \wedge \neg\, wait' \wedge (\mathbf{R}(Q_{pre} \vdash Q_{post}))^t_f \end{array} \right\} \end{array} \right\}
\end{aligned}
$$

[Assumption 2]

$$
\begin{aligned}
&= \{(tr' - tr, ref') \mid okay \wedge wait' \wedge (P)^t_f\} \\
&\cup \{(tr' - tr, ref' \cup \{\checkmark\}) \mid okay \wedge wait' \wedge (P)^t_f\} \\
&\cup \left\{ \begin{array}{l} (s \frown t, X) \mid \\ \qquad s \in \{(tr' - tr) \mid okay \wedge \neg\, wait' \wedge (P)^t_f\} \\ \qquad \wedge (t, X) \in \{(tr' - tr, ref') \mid okay \wedge (Q)^t_f\} \end{array} \right\} \\
&\cup \left\{ \begin{array}{l} (s \frown t, X) \mid \\ \qquad s \in \{(tr' - tr) \mid okay \wedge \neg\, wait' \wedge (P)^t_f\} \\ \qquad \wedge (t, X) \in \left\{ \begin{array}{l} (tr' - tr, ref' \cup \{\checkmark\}) \mid \\ \qquad okay \wedge wait' \wedge (Q)^t_f \end{array} \right\} \end{array} \right\} \\
&\cup \left\{ \begin{array}{l} (s \frown t, X) \mid \\ \qquad s \in \{(tr' - tr) \mid okay \wedge \neg\, wait' \wedge (P)^t_f\} \\ \qquad \wedge (t, X) \in \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref') \mid \\ \qquad okay \wedge \neg\, wait' \wedge (Q)^t_f \end{array} \right\} \end{array} \right\} \\
&\cup \left\{ \begin{array}{l} (s \frown t, X) \mid \\ \qquad s \in \{(tr' - tr) \mid okay \wedge \neg\, wait' \wedge (P)^t_f\} \\ \qquad \wedge (t, X) \in \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref' \cup \{\checkmark\}) \mid \\ \qquad okay \wedge \neg\, wait' \wedge (Q)^t_f \end{array} \right\} \end{array} \right\}
\end{aligned}
$$

[PC]

$$
\begin{aligned}
&= \{(tr' - tr, ref') \mid okay \land \neg\, wait \land okay' \land wait' \land P\}\\
&\quad \cup \{(tr' - tr, ref' \cup \{\checkmark\}) \mid okay \land \neg\, wait \land okay' \land wait' \land P\}\\
&\quad \cup \left\{\begin{array}{l} (s \frown t, X) \mid \\ \qquad s \in \{(tr' - tr) \mid okay \land \neg\, wait \land okay' \land \neg\, wait' \land P\} \\ \qquad \land (t, X) \in \{(tr' - tr, ref') \mid okay \land \neg\, wait \land okay' \land Q\} \end{array}\right\}\\
&\quad \cup \left\{\begin{array}{l} (s \frown t, X) \mid \\ \qquad s \in \{(tr' - tr) \mid okay \land \neg\, wait \land okay' \land \neg\, wait' \land P\} \\ \qquad \land (t, X) \in \left\{\begin{array}{l} (tr' - tr, ref' \cup \{\checkmark\}) \mid \\ \qquad okay \land \neg\, wait \land okay' \land wait' \land Q \end{array}\right\} \end{array}\right\}\\
&\quad \cup \left\{\begin{array}{l} (s \frown t, X) \mid \\ \qquad s \in \{(tr' - tr) \mid okay \land \neg\, wait \land okay' \land \neg\, wait' \land P\} \\ \qquad \land (t, X) \in \left\{\begin{array}{l} ((tr' - tr) \frown \langle\checkmark\rangle, ref') \mid \\ \qquad okay \land \neg\, wait \land okay' \land \neg\, wait' \land Q \end{array}\right\} \end{array}\right\}\\
&\quad \cup \left\{\begin{array}{l} (s \frown t, X) \mid \\ \qquad s \in \{(tr' - tr) \mid okay \land \neg\, wait \land okay' \land \neg\, wait' \land P\} \\ \qquad \land (t, X) \in \left\{\begin{array}{l} ((tr' - tr) \frown \langle\checkmark\rangle, ref' \cup \{\checkmark\}) \mid \\ \qquad okay \land \neg\, wait \land okay' \land \neg\, wait' \land Q \end{array}\right\} \end{array}\right\}
\end{aligned}
$$

$$[A^n]$$

$$
\begin{aligned}
&= \{(tr' - tr, ref') \mid (P)^n \land wait'\}\\
&\quad \cup \{(tr' - tr, ref' \cup \{\checkmark\}) \mid (P)^n \land wait'\}\\
&\quad \cup \left\{\begin{array}{l} (s \frown t, X) \mid \\ \qquad s \in \{(tr' - tr) \mid \neg\, wait' \land (P)^n\} \\ \qquad \land (t, X) \in \{(tr' - tr, ref') \mid (Q)^n\} \end{array}\right\}\\
&\quad \cup \left\{\begin{array}{l} (s \frown t, X) \mid \\ \qquad s \in \{(tr' - tr) \mid \neg\, wait' \land (P)^n\} \\ \qquad \land (t, X) \in \{(tr' - tr, ref' \cup \{\checkmark\}) \mid (Q)^n \land wait'\} \end{array}\right\}\\
&\quad \cup \left\{\begin{array}{l} (s \frown t, X) \mid \\ \qquad s \in \{(tr' - tr) \mid \neg\, wait' \land (P)^n\} \\ \qquad \land (t, X) \in \{((tr' - tr) \frown \langle\checkmark\rangle, ref') \mid \neg\, wait' \land (Q)^n\} \end{array}\right\}\\
&\quad \cup \left\{\begin{array}{l} (s \frown t, X) \mid \\ \qquad s \in \{(tr' - tr) \mid \neg\, wait' \land (P)^n\} \\ \qquad \land (t, X) \in \{((tr' - tr) \frown \langle\checkmark\rangle, ref' \cup \{\checkmark\}) \mid \neg\, wait' \land (Q)^n\} \end{array}\right\}
\end{aligned}
$$

$$[A^t]$$

$$
\begin{array}{l}
= \{(tr' - tr, ref') \mid (P)^n \land wait'\} \\
\quad \cup \{(tr' - tr, ref' \cup \{\checkmark\}) \mid (P)^n \land wait'\} \\
\quad \cup \left\{ \begin{array}{l} (s \frown t, X) \mid \\ \qquad s \in \{(tr' - tr) \mid (P)^t\} \\ \qquad \land (t, X) \in \{(tr' - tr, ref') \mid (Q)^n\} \end{array} \right\} \\
\quad \cup \left\{ \begin{array}{l} (s \frown t, X) \mid \\ \qquad s \in \{(tr' - tr) \mid (P)^t\} \\ \qquad \land (t, X) \in \{(tr' - tr, ref' \cup \{\checkmark\}) \mid (Q)^n \land wait'\} \end{array} \right\} \\
\quad \cup \left\{ \begin{array}{l} (s \frown t, X) \mid \\ \qquad s \in \{(tr' - tr) \mid (P)^t\} \\ \qquad \land (t, X) \in \{((tr' - tr) \frown \langle \checkmark \rangle, ref') \mid (Q)^t\} \end{array} \right\} \\
\quad \cup \left\{ \begin{array}{l} (s \frown t, X) \mid \\ \qquad s \in \{(tr' - tr) \mid (P)^t\} \\ \qquad \land (t, X) \in \{((tr' - tr) \frown \langle \checkmark \rangle, ref' \cup \{\checkmark\}) \mid (Q)^t\} \end{array} \right\}
\end{array}
$$

[ST, PC and SC]

$$
\begin{array}{l}
= \{(tr' - tr, ref') \mid (P)^n \land wait'\} \\
\quad \cup \{(tr' - tr, ref' \cup \{\checkmark\}) \mid (P)^n \land wait'\} \\
\quad \cup \\
\quad \left\{ \begin{array}{l} (s \frown t, X) \mid \\ \qquad s \in \{(tr' - tr) \mid (P)^t\} \\ \qquad \land (t, X) \in \left\{ \begin{array}{l} (tr' - tr, ref') \mid (Q)^n\} \\ \cup \{(tr' - tr, ref' \cup \{\checkmark\}) \mid (Q)^n \land wait'\} \\ \cup \{((tr' - tr) \frown \langle \checkmark \rangle, ref') \mid (Q)^t\} \\ \cup \{((tr' - tr) \frown \langle \checkmark \rangle, ref' \cup \{\checkmark\}) \mid (Q)^t \end{array} \right\} \end{array} \right\}
\end{array}
$$

[SC and ST $(ref, ref' : \operatorname{seq} \Sigma$ and $\checkmark \notin \Sigma)$]

$$
\begin{array}{l}
= \left\{ \begin{array}{l} (s, X) \mid \\ \qquad s \in \Sigma^* \\ \qquad \land (s, X \cup \{\checkmark\}) \in \{(tr' - tr, ref' \cup \{\checkmark\}) \mid (P)^n \land wait'\} \end{array} \right\} \\
\quad \cup \\
\quad \left\{ \begin{array}{l} (s \frown t, X) \mid \\ \qquad s \in \{(tr' - tr) \mid (P)^t\} \\ \qquad \land (t, X) \in \left\{ \begin{array}{l} (tr' - tr, ref') \mid (Q)^n\} \\ \cup \{(tr' - tr, ref' \cup \{\checkmark\}) \mid (Q)^n \land wait'\} \\ \cup \{((tr' - tr) \frown \langle \checkmark \rangle, ref') \mid (Q)^t\} \\ \cup \{((tr' - tr) \frown \langle \checkmark \rangle, ref' \cup \{\checkmark\}) \mid (Q)^t \end{array} \right\} \end{array} \right\}
\end{array}
$$

[SC and ST $(ref, ref' : \operatorname{seq} \Sigma$ and $\checkmark \notin \Sigma)$]

$$= \left\{ \begin{array}{l} (s, X) \mid \\ \qquad s \in \Sigma^* \\ \qquad \wedge\ (s, X \cup \{\checkmark\}) \in \left\{ \begin{array}{l} \{(tr' - tr, ref') \mid (P)^n\} \\ \cup \{(tr' - tr, ref' \cup \{\checkmark\}) \mid (P)^n \wedge wait'\} \end{array} \right\} \end{array} \right\}$$

$$\cup$$

$$\left\{ \begin{array}{l} (s \frown t, X) \mid \\ \qquad s \in \{(tr' - tr) \mid (P)^t\} \\ \qquad \wedge\ (t, X) \in \left\{ \begin{array}{l} \{(tr' - tr, ref') \mid (Q)^n\} \\ \cup \{(tr' - tr, ref' \cup \{\checkmark\}) \mid (Q)^n \wedge wait'\} \\ \cup \{((tr' - tr) \frown \langle \checkmark \rangle, ref') \mid (Q)^t\} \\ \cup \{((tr' - tr) \frown \langle \checkmark \rangle, ref' \cup \{\checkmark\}) \mid (Q)^t \end{array} \right\} \end{array} \right\}$$

[SC and ST $(tr, tr' : \operatorname{seq} \Sigma$ and $\checkmark \notin \Sigma)$]

$$= \left\{ \begin{array}{l} (s, X) \mid \\ \qquad s \in \Sigma^* \\ \qquad \wedge\ (s, X \cup \{\checkmark\}) \in \left\{ \begin{array}{l} \{(tr' - tr, ref') \mid (P)^n\} \\ \cup \{(tr' - tr, ref' \cup \{\checkmark\}) \mid (P)^n \wedge wait'\} \\ \cup \{((tr' - tr) \frown \langle \checkmark \rangle, ref') \mid (P)^t\} \\ \cup \{((tr' - tr) \frown \langle \checkmark \rangle, ref' \cup \{\checkmark\}) \mid (P)^t \end{array} \right\} \end{array} \right\}$$

$$\cup$$

$$\left\{ \begin{array}{l} (s \frown t, X) \mid \\ \qquad s \in \{(tr' - tr) \mid (P)^t\} \\ \qquad \wedge\ (t, X) \in \left\{ \begin{array}{l} \{(tr' - tr, ref') \mid (Q)^n\} \\ \cup \{(tr' - tr, ref' \cup \{\checkmark\}) \mid (Q)^n \wedge wait'\} \\ \cup \{((tr' - tr) \frown \langle \checkmark \rangle, ref') \mid (Q)^t\} \\ \cup \{((tr' - tr) \frown \langle \checkmark \rangle, ref' \cup \{\checkmark\}) \mid (Q)^t \end{array} \right\} \end{array} \right\}$$

[SC and ST $(tr, tr' : \operatorname{seq} \Sigma$ and $\checkmark \notin \Sigma)$]

$$
\begin{aligned}
&= \left\{ \begin{array}{l} (s, X) \mid \\ \qquad s \in \Sigma^* \\ \qquad \wedge\, (s, X \cup \{\checkmark\}) \in \left\{ \begin{array}{l} \{(tr' - tr, ref') \mid (P)^n\} \\ \cup \{(tr' - tr, ref' \cup \{\checkmark\}) \mid (P)^n \wedge wait'\} \\ \cup \{((tr' - tr) \frown \langle\checkmark\rangle, ref') \mid (P)^t\} \\ \cup \{((tr' - tr) \frown \langle\checkmark\rangle, ref' \cup \{\checkmark\}) \mid (P)^t \end{array} \right\} \end{array} \right\} \\
&\cup \\
&\quad \left\{ \begin{array}{l} (s \frown t, X) \mid \\ \qquad s \frown \langle\checkmark\rangle \in \\ \qquad\qquad \{tr' - tr \mid (P)^n\} \cup \{(tr' - tr) \frown \langle\checkmark\rangle \mid (P)^t\} \\ \qquad \wedge\, (t, X) \in \left\{ \begin{array}{l} \{(tr' - tr, ref') \mid (Q)^n\} \\ \cup \{(tr' - tr, ref' \cup \{\checkmark\}) \mid (Q)^n \wedge wait'\} \\ \cup \{((tr' - tr) \frown \langle\checkmark\rangle, ref') \mid (Q)^t\} \\ \cup \{((tr' - tr) \frown \langle\checkmark\rangle, ref' \cup \{\checkmark\}) \mid (Q)^t \end{array} \right\} \end{array} \right\} \\
&\hfill [traces^{\mathcal{UTP}} \text{ and } failures^{\mathcal{UTP}}] \\
&= \{(s, X) \mid s \in \Sigma^* \wedge (s, X \cup \{\checkmark\}) \in failures^{\mathcal{UTP}}(P)\} \hfill [\text{Theorem 5.1}] \\
&\quad \cup \{(s \frown t, X) \mid s \frown \langle\checkmark\rangle \in traces^{\mathcal{UTP}}(P) \wedge (t, X) \in failures^{\mathcal{UTP}}(Q)\} \\
&= \{(s, X) \mid s \in \Sigma^* \wedge (s, X \cup \{\checkmark\}) \in failures^{\mathcal{UTP}}(P)\} \hfill [\text{IH}] \\
&\quad \cup \{(s \frown t, X) \mid s \frown \langle\checkmark\rangle \in traces(\Upsilon(P)) \wedge (t, X) \in failures^{\mathcal{UTP}}(Q)\} \\
&= \{(s, X) \mid s \in \Sigma^* \wedge (s, X \cup \{\checkmark\}) \in failures(\Upsilon(P))\} \hfill [failures] \\
&\quad \cup \{(s \frown t, X) \mid s \frown \langle\checkmark\rangle \in traces(\Upsilon(P)) \wedge (t, X) \in failures(\Upsilon(Q))\} \\
&= failures(\Upsilon(P);\ \Upsilon(Q)) \hfill [\Upsilon] \\
&= failures(\Upsilon(P;\ Q))
\end{aligned}
$$

Theorem J.35

$$failures^{\mathcal{UTP}}(\text{⨾}\ x : S \bullet A) = failures(\Upsilon(\text{⨾}\ x : S \bullet A))$$

provided

1. $\forall\, i : S \bullet A[v_i/x]$ *is* $\boldsymbol{R}$
2. $\forall\, i : S \bullet A[v_i/x]$ *is divergence-free*

Inductive Hypothesis (A):

$$\forall\, i : S \bullet failures^{\mathcal{UTP}}(A[v_i/x]) = failures(\Upsilon(A)[v_i/x])$$

Proof. By induction on S

Base Case. $S = \langle\rangle$
Proof.

$$\begin{aligned}
& \mathit{failures}^{\mathcal{UTP}}(\mathbin{;}\ x : S \bullet A) && \text{[Assumption]}\\
&= \mathit{failures}^{\mathcal{UTP}}(\mathbin{;}\ x : \langle\rangle \bullet A) && \text{[Property of ;]}\\
&= \mathit{failures}^{\mathcal{UTP}}(\mathit{Skip}) && \text{[Theorem J.21]}\\
&= \mathit{failures}(\Upsilon(\mathit{Skip})) && \text{[Property of ;]}\\
&= \mathit{failures}(\Upsilon(\mathbin{;}\ x : \langle\rangle \bullet A)) && \text{[Assumption]}\\
&= \mathit{failures}(\Upsilon(\mathbin{;}\ x : S \bullet A))
\end{aligned}$$

Inductive Hypothesis (S):

$$\mathit{failures}^{\mathcal{UTP}}(\mathbin{;}\ x : S \bullet A) = \mathit{failures}(\Upsilon(\mathbin{;}\ x : S \bullet A))$$

Inductive Step

$$\mathit{failures}^{\mathcal{UTP}}(\mathbin{;}\ x : S \cup \{v_i\} \bullet A) = \mathit{failures}(\Upsilon(\mathbin{;}\ x : S \cup \{v_i\} \bullet A))$$

Proof.

$$\begin{aligned}
& \mathit{failures}^{\mathcal{UTP}}(\mathbin{;}\ x : S \bullet A) && [\mathbin{;}]\\
&= \mathit{failures}^{\mathcal{UTP}}(A[\mathit{head}(s)/x];\ (\mathbin{;}\ x : \mathit{tail}(S) \bullet A))\\
& && \text{[Theorem J.34 (Provisos, IH-}A\text{ and IH-}S\text{)]}\\
&= \mathit{failures}(\Upsilon(A[\mathit{head}(v_i)/x];\ (\mathbin{;}\ x : \mathit{tail}(S) \bullet A))) && [\mathbin{;}]\\
&= \mathit{failures}(\Upsilon(\mathbin{;}\ x : S \bullet A))
\end{aligned}$$

Theorem J.36

$$\begin{aligned}
& \mathit{failures}^{\mathcal{UTP}}(P \llbracket \mathit{ns}_1 \mid \mathit{cs} \mid \mathit{ns}_2 \rrbracket Q)\\
& =\\
& \mathit{failures}(\Upsilon(P \llbracket \mathit{ns}_1 \mid \mathit{cs} \mid \mathit{ns}_2 \rrbracket Q))
\end{aligned}$$

provided

1. *P and Q are divergence-free*

Inductive Hypothesis:

$$\begin{aligned}
& \mathit{failures}^{\mathcal{UTP}}(P) = \mathit{failures}(\Upsilon(P))\\
& \text{and}\\
& \mathit{failures}^{\mathcal{UTP}}(Q) = \mathit{failures}(\Upsilon(Q))
\end{aligned}$$

Proof.

$$
\begin{array}{ll}
\mathit{failures}^{\mathcal{UTP}}(P \llbracket ns_1 \mid cs \mid ns_2 \rrbracket Q) & [\mathit{failures}^{\mathcal{UTP}}] \\
= \{(tr' - tr, ref') \mid (P \llbracket ns_1 \mid cs \mid ns_2 \rrbracket Q)^n\} & [A^t] \\
\quad \cup \{(tr' - tr, ref' \cup \{\checkmark\}) \mid (P \llbracket ns_1 \mid cs \mid ns_2 \rrbracket Q)^n \wedge wait'\} & \\
\quad \cup \{((tr' - tr) \frown \langle \checkmark \rangle, ref') \mid (P \llbracket ns_1 \mid cs \mid ns_2 \rrbracket Q)^t\} & \\
\quad \cup \{((tr' - tr) \frown \langle \checkmark \rangle, ref' \cup \{\checkmark\}) \mid (P \llbracket ns_1 \mid cs \mid ns_2 \rrbracket Q)^t\} & \\
= \{(tr' - tr, ref') \mid (P \llbracket ns_1 \mid cs \mid ns_2 \rrbracket Q)^n\} & [A^n] \\
\quad \cup \{(tr' - tr, ref' \cup \{\checkmark\}) \mid (P \llbracket ns_1 \mid cs \mid ns_2 \rrbracket Q)^n \wedge wait'\} & \\
\quad \cup \{((tr' - tr) \frown \langle \checkmark \rangle, ref') \mid \neg\, wait' \wedge (P \llbracket ns_1 \mid cs \mid ns_2 \rrbracket Q)^n\} & \\
\quad \cup \{((tr' - tr) \frown \langle \checkmark \rangle, ref' \cup \{\checkmark\}) \mid \neg\, wait' \wedge (P \llbracket ns_1 \mid cs \mid ns_2 \rrbracket Q)^n\} & \\
= \{(tr' - tr, ref') \mid okay \wedge \neg\, wait \wedge okay' \wedge (P \llbracket ns_1 \mid cs \mid ns_2 \rrbracket Q)\} & [\mathrm{PC}] \\
\quad \cup \left\{ \begin{array}{l} (tr' - tr, ref' \cup \{\checkmark\}) \\ \mid okay \wedge \neg\, wait \wedge okay' \wedge wait' \wedge (P \llbracket ns_1 \mid cs \mid ns_2 \rrbracket Q) \end{array} \right\} & \\
\quad \cup \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref') \\ \mid okay \wedge \neg\, wait \wedge okay' \wedge \neg\, wait' \wedge (P \llbracket ns_1 \mid cs \mid ns_2 \rrbracket Q) \end{array} \right\} & \\
\quad \cup \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref' \cup \{\checkmark\}) \mid \\ okay \wedge \neg\, wait \wedge okay' \wedge \neg\, wait' \wedge (P \llbracket ns_1 \mid cs \mid ns_2 \rrbracket Q) \end{array} \right\} & \\
= \{(tr' - tr, ref') \mid okay \wedge (P \llbracket ns_1 \mid cs \mid ns_2 \rrbracket Q)^t_f\} & \\
\quad \cup \left\{ \begin{array}{l} (tr' - tr, ref' \cup \{\checkmark\}) \\ \mid okay \wedge wait' \wedge (P \llbracket ns_1 \mid cs \mid ns_2 \rrbracket Q)^t_f \end{array} \right\} & \\
\quad \cup \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref') \\ \mid okay \wedge \neg\, wait' \wedge (P \llbracket ns_1 \mid cs \mid ns_2 \rrbracket Q)^t_f \end{array} \right\} & \\
\quad \cup \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref' \cup \{\checkmark\}) \mid \\ okay \wedge \neg\, wait' \wedge (P \llbracket ns_1 \mid cs \mid ns_2 \rrbracket Q)^t_f \end{array} \right\} & \\
& [\text{Lemma J.28 (Assumptions)}]
\end{array}
$$

$$
= \left\{ \begin{array}{l}
(tr' - tr, ref') \\
\mid okay \land \mathbf{CSP1} \left(\begin{array}{l}
\mathbf{R1} \left(\begin{array}{l} \exists\, 1.tr', 2.tr' \bullet (P^f_f;\ 1.tr' = tr) \\ \qquad \land (Q_f;\ 2.tr' = tr) \\ \qquad \land 1.tr' \upharpoonright cs = 2.tr' \upharpoonright cs \end{array} \right) \\
\lor \mathbf{R1} \left(\begin{array}{l} \exists\, 1.tr', 2.tr' \bullet (P_f;\ 1.tr' = tr) \\ \qquad \land (Q^f_f;\ 2.tr' = tr) \\ \qquad \land 1.tr' \upharpoonright cs = 2.tr' \upharpoonright cs \end{array} \right) \\
\lor \left(\left(\begin{array}{l} (P^t_f;\ U1(out\alpha\, P)) \\ \land (Q^t_f;\ U2(out\alpha\, Q)) \end{array} \right)_{+\{v,tr\}} ;\ M_{\|_{cs}} \right)
\end{array} \right)
\end{array} \right\}
$$

$$
\cup \left\{ \begin{array}{l}
(tr' - tr, ref' \cup \{\checkmark\}) \\
\mid okay \land wait' \land \\
\mathbf{CSP1} \left(\begin{array}{l}
\mathbf{R1} \left(\begin{array}{l} \exists\, 1.tr', 2.tr' \bullet (P^f_f;\ 1.tr' = tr) \\ \qquad \land (Q_f;\ 2.tr' = tr) \\ \qquad \land 1.tr' \upharpoonright cs = 2.tr' \upharpoonright cs \end{array} \right) \\
\lor \mathbf{R1} \left(\begin{array}{l} \exists\, 1.tr', 2.tr' \bullet (P_f;\ 1.tr' = tr) \\ \qquad \land (Q^f_f;\ 2.tr' = tr) \\ \qquad \land 1.tr' \upharpoonright cs = 2.tr' \upharpoonright cs \end{array} \right) \\
\lor \left(\left(\begin{array}{l} (P^t_f;\ U1(out\alpha\, P)) \\ \land (Q^t_f;\ U2(out\alpha\, Q)) \end{array} \right)_{+\{v,tr\}} ;\ M_{\|_{cs}} \right)
\end{array} \right)
\end{array} \right\}
$$

$$
\cup \left\{ \begin{array}{l}
((tr' - tr) \frown \langle \checkmark \rangle, ref') \\
\mid okay \land \neg\ wait' \land \\
\mathbf{CSP1} \left(\begin{array}{l}
\mathbf{R1} \left(\begin{array}{l} \exists\, 1.tr', 2.tr' \bullet (P^f_f;\ 1.tr' = tr) \\ \qquad \land (Q_f;\ 2.tr' = tr) \\ \qquad \land 1.tr' \upharpoonright cs = 2.tr' \upharpoonright cs \end{array} \right) \\
\lor \mathbf{R1} \left(\begin{array}{l} \exists\, 1.tr', 2.tr' \bullet (P_f;\ 1.tr' = tr) \\ \qquad \land (Q^f_f;\ 2.tr' = tr) \\ \qquad \land 1.tr' \upharpoonright cs = 2.tr' \upharpoonright cs \end{array} \right) \\
\lor \left(\left(\begin{array}{l} (P^t_f;\ U1(out\alpha\, P)) \\ \land (Q^t_f;\ U2(out\alpha\, Q)) \end{array} \right)_{+\{v,tr\}} ;\ M_{\|_{cs}} \right)
\end{array} \right)
\end{array} \right\}
$$

$$
\cup \left\{ \begin{array}{l}
((tr' - tr) \frown \langle \checkmark \rangle, ref' \cup \{\checkmark\}) \mid \\
okay \land \neg\ wait' \land \\
\mathbf{CSP1} \left(\begin{array}{l}
\mathbf{R1} \left(\begin{array}{l} \exists\, 1.tr', 2.tr' \bullet (P^f_f;\ 1.tr' = tr) \\ \qquad \land (Q_f;\ 2.tr' = tr) \\ \qquad \land 1.tr' \upharpoonright cs = 2.tr' \upharpoonright cs \end{array} \right) \\
\lor \mathbf{R1} \left(\begin{array}{l} \exists\, 1.tr', 2.tr' \bullet (P_f;\ 1.tr' = tr) \\ \qquad \land (Q^f_f;\ 2.tr' = tr) \\ \qquad \land 1.tr' \upharpoonright cs = 2.tr' \upharpoonright cs \end{array} \right) \\
\lor \left(\left(\begin{array}{l} (P^t_f;\ U1(out\alpha\, P)) \\ \land (Q^t_f;\ U2(out\alpha\, Q)) \end{array} \right)_{+\{v,tr\}} ;\ M_{\|_{cs}} \right)
\end{array} \right)
\end{array} \right\}
$$

[Lemma J.4]

$$= \left\{ \begin{array}{l} (tr' - tr, ref') \\ \mid okay \wedge \left(\begin{array}{l} \mathbf{R1} \left(\begin{array}{r} \exists\, 1.tr', 2.tr' \bullet (P_f^f;\ 1.tr' = tr) \\ \wedge\, (Q_f;\ 2.tr' = tr) \\ \wedge\, 1.tr' \upharpoonright cs = 2.tr' \upharpoonright cs \end{array} \right) \\ \vee\, \mathbf{R1} \left(\begin{array}{r} \exists\, 1.tr', 2.tr' \bullet (P_f;\ 1.tr' = tr) \\ \wedge\, (Q_f^f;\ 2.tr' = tr) \\ \wedge\, 1.tr' \upharpoonright cs = 2.tr' \upharpoonright cs \end{array} \right) \\ \vee \left(\left(\begin{array}{l} (P_f^t;\ U1(out\alpha\, P)) \\ \wedge\, (Q_f^t;\ U2(out\alpha\, Q)) \end{array} \right)_{+\{v,tr\}} ;\ M_{\parallel_{cs}} \right) \end{array} \right) \end{array} \right\}$$

$$\cup \left\{ \begin{array}{l} (tr' - tr, ref' \cup \{\checkmark\}) \\ \mid okay \wedge wait' \wedge \\ \quad \left(\begin{array}{l} \mathbf{R1} \left(\begin{array}{r} \exists\, 1.tr', 2.tr' \bullet (P_f^f;\ 1.tr' = tr) \\ \wedge\, (Q_f;\ 2.tr' = tr) \\ \wedge\, 1.tr' \upharpoonright cs = 2.tr' \upharpoonright cs \end{array} \right) \\ \vee\, \mathbf{R1} \left(\begin{array}{r} \exists\, 1.tr', 2.tr' \bullet (P_f;\ 1.tr' = tr) \\ \wedge\, (Q_f^f;\ 2.tr' = tr) \\ \wedge\, 1.tr' \upharpoonright cs = 2.tr' \upharpoonright cs \end{array} \right) \\ \vee \left(\left(\begin{array}{l} (P_f^t;\ U1(out\alpha\, P)) \\ \wedge\, (Q_f^t;\ U2(out\alpha\, Q)) \end{array} \right)_{+\{v,tr\}} ;\ M_{\parallel_{cs}} \right) \end{array} \right) \end{array} \right\}$$

$$\cup \left\{ \begin{array}{l} ((tr' - tr) ^\frown \langle \checkmark \rangle, ref') \\ \mid okay \wedge \neg\, wait' \wedge \\ \quad \left(\begin{array}{l} \mathbf{R1} \left(\begin{array}{r} \exists\, 1.tr', 2.tr' \bullet (P_f^f;\ 1.tr' = tr) \\ \wedge\, (Q_f;\ 2.tr' = tr) \\ \wedge\, 1.tr' \upharpoonright cs = 2.tr' \upharpoonright cs \end{array} \right) \\ \vee\, \mathbf{R1} \left(\begin{array}{r} \exists\, 1.tr', 2.tr' \bullet (P_f;\ 1.tr' = tr) \\ \wedge\, (Q_f^f;\ 2.tr' = tr) \\ \wedge\, 1.tr' \upharpoonright cs = 2.tr' \upharpoonright cs \end{array} \right) \\ \vee \left(\left(\begin{array}{l} (P_f^t;\ U1(out\alpha\, P)) \\ \wedge\, (Q_f^t;\ U2(out\alpha\, Q)) \end{array} \right)_{+\{v,tr\}} ;\ M_{\parallel_{cs}} \right) \end{array} \right) \end{array} \right\}$$

$$\cup \left\{ \begin{array}{l} ((tr' - tr) ^\frown \langle \checkmark \rangle, ref' \cup \{\checkmark\}) \mid \\ okay \wedge \neg\, wait' \wedge \\ \quad \left(\begin{array}{l} \mathbf{R1} \left(\begin{array}{r} \exists\, 1.tr', 2.tr' \bullet (P_f^f;\ 1.tr' = tr) \\ \wedge\, (Q_f;\ 2.tr' = tr) \\ \wedge\, 1.tr' \upharpoonright cs = 2.tr' \upharpoonright cs \end{array} \right) \\ \vee\, \mathbf{R1} \left(\begin{array}{r} \exists\, 1.tr', 2.tr' \bullet (P_f;\ 1.tr' = tr) \\ \wedge\, (Q_f^f;\ 2.tr' = tr) \\ \wedge\, 1.tr' \upharpoonright cs = 2.tr' \upharpoonright cs \end{array} \right) \\ \vee \left(\left(\begin{array}{l} (P_f^t;\ U1(out\alpha\, P)) \\ \wedge\, (Q_f^t;\ U2(out\alpha\, Q)) \end{array} \right)_{+\{v,tr\}} ;\ M_{\parallel_{cs}} \right) \end{array} \right) \end{array} \right.$$

[Assumption 1]

$$= \left\{ \begin{array}{l} (tr' - tr, ref') \\ \mid okay \wedge \left(\begin{array}{l} \mathbf{R1}\left(\begin{array}{l} \exists\, 1.tr', 2.tr' \bullet (false;\ 1.tr' = tr) \\ \qquad \wedge (Q_f;\ 2.tr' = tr) \\ \qquad \wedge 1.tr' \upharpoonright cs = 2.tr' \upharpoonright cs \end{array} \right) \\ \vee\, \mathbf{R1}\left(\begin{array}{l} \exists\, 1.tr', 2.tr' \bullet (P_f;\ 1.tr' = tr) \\ \qquad \wedge (false;\ 2.tr' = tr) \\ \qquad \wedge 1.tr' \upharpoonright cs = 2.tr' \upharpoonright cs \end{array} \right) \\ \vee \left(\left(\begin{array}{l} (P_f^t;\ U1(out\alpha\, P)) \\ \wedge\, (Q_f^t;\ U2(out\alpha\, Q)) \end{array} \right)_{+\{v,tr\}} ;\ M_{\parallel_{cs}} \right) \end{array} \right) \end{array} \right\}$$

$$\cup \left\{ \begin{array}{l} (tr' - tr, ref' \cup \{\checkmark\}) \\ \mid okay \wedge wait' \wedge \\ \qquad \left(\begin{array}{l} \mathbf{R1}\left(\begin{array}{l} \exists\, 1.tr', 2.tr' \bullet (false;\ 1.tr' = tr) \\ \qquad \wedge (Q_f;\ 2.tr' = tr) \\ \qquad \wedge 1.tr' \upharpoonright cs = 2.tr' \upharpoonright cs \end{array} \right) \\ \vee\, \mathbf{R1}\left(\begin{array}{l} \exists\, 1.tr', 2.tr' \bullet (P_f;\ 1.tr' = tr) \\ \qquad \wedge (false;\ 2.tr' = tr) \\ \qquad \wedge 1.tr' \upharpoonright cs = 2.tr' \upharpoonright cs \end{array} \right) \\ \vee \left(\left(\begin{array}{l} (P_f^t;\ U1(out\alpha\, P)) \\ \wedge\, (Q_f^t;\ U2(out\alpha\, Q)) \end{array} \right)_{+\{v,tr\}} ;\ M_{\parallel_{cs}} \right) \end{array} \right) \end{array} \right\}$$

$$\cup \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref') \\ \mid okay \wedge \neg\, wait' \wedge \\ \qquad \left(\begin{array}{l} \mathbf{R1}\left(\begin{array}{l} \exists\, 1.tr', 2.tr' \bullet (false;\ 1.tr' = tr) \\ \qquad \wedge (Q_f;\ 2.tr' = tr) \\ \qquad \wedge 1.tr' \upharpoonright cs = 2.tr' \upharpoonright cs \end{array} \right) \\ \vee\, \mathbf{R1}\left(\begin{array}{l} \exists\, 1.tr', 2.tr' \bullet (P_f;\ 1.tr' = tr) \\ \qquad \wedge (false;\ 2.tr' = tr) \\ \qquad \wedge 1.tr' \upharpoonright cs = 2.tr' \upharpoonright cs \end{array} \right) \\ \vee \left(\left(\begin{array}{l} (P_f^t;\ U1(out\alpha\, P)) \\ \wedge\, (Q_f^t;\ U2(out\alpha\, Q)) \end{array} \right)_{+\{v,tr\}} ;\ M_{\parallel_{cs}} \right) \end{array} \right) \end{array} \right\}$$

$$\cup \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref' \cup \{\checkmark\}) \mid \\ okay \wedge \neg\, wait' \wedge \\ \qquad \left(\begin{array}{l} \mathbf{R1}\left(\begin{array}{l} \exists\, 1.tr', 2.tr' \bullet (false;\ 1.tr' = tr) \\ \qquad \wedge (Q_f;\ 2.tr' = tr) \\ \qquad \wedge 1.tr' \upharpoonright cs = 2.tr' \upharpoonright cs \end{array} \right) \\ \vee\, \mathbf{R1}\left(\begin{array}{l} \exists\, 1.tr', 2.tr' \bullet (P_f;\ 1.tr' = tr) \\ \qquad \wedge (false;\ 2.tr' = tr) \\ \qquad \wedge 1.tr' \upharpoonright cs = 2.tr' \upharpoonright cs \end{array} \right) \\ \vee \left(\left(\begin{array}{l} (P_f^t;\ U1(out\alpha\, P)) \\ \wedge\, (Q_f^t;\ U2(out\alpha\, Q)) \end{array} \right)_{+\{v,tr\}} ;\ M_{\parallel_{cs}} \right) \end{array} \right) \end{array} \right\}$$

[Sequence and PC]

$$
\begin{array}{l}
= \left\{ \begin{array}{l} (tr' - tr, ref') \\ \mid okay \land \left(\left(\begin{array}{l} (P^t_f;\ U1(out\alpha\, P)) \\ \land\, (Q^t_f;\ U2(out\alpha\, Q)) \end{array} \right)_{+\{v,tr\}} ;\ M_{\parallel_{cs}} \right) \end{array} \right\} \\
\cup \left\{ \begin{array}{l} (tr' - tr, ref' \cup \{\checkmark\}) \\ \mid okay \land wait' \land \\ \qquad \left(\left(\begin{array}{l} (P^t_f;\ U1(out\alpha\, P)) \\ \land\, (Q^t_f;\ U2(out\alpha\, Q)) \end{array} \right)_{+\{v,tr\}} ;\ M_{\parallel_{cs}} \right) \end{array} \right\} \\
\cup \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref') \\ \mid okay \land \neg\, wait' \land \\ \qquad \left(\left(\begin{array}{l} (P^t_f;\ U1(out\alpha\, P)) \\ \land\, (Q^t_f;\ U2(out\alpha\, Q)) \end{array} \right)_{+\{v,tr\}} ;\ M_{\parallel_{cs}} \right) \end{array} \right\} \\
\cup \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref' \cup \{\checkmark\}) \mid \\ okay \land \neg\, wait' \land \\ \qquad \left(\left(\begin{array}{l} (P^t_f;\ U1(out\alpha\, P)) \\ \land\, (Q^t_f;\ U2(out\alpha\, Q)) \end{array} \right)_{+\{v,tr\}} ;\ M_{\parallel_{cs}} \right) \end{array} \right\}
\end{array}
$$

$[M_{\parallel_{cs}}]$

$$
\begin{array}{l}
= \left\{ \begin{array}{l}
(tr' - tr, ref') \\
\mid okay \wedge \left(\begin{array}{l}
\left(\begin{array}{l} (P^t_f;\ U1(out\alpha\, P)) \\ \wedge\, (Q^t_f;\ U2(out\alpha\, Q)) \end{array} \right)_{+\{v,tr\}} ; \\
\left(\begin{array}{l}
tr' - tr \in (1.tr - tr \parallel_{cs} 2.tr - tr) \\
\wedge\, 1.tr \upharpoonright cs = 2.tr \upharpoonright cs \\
\wedge \left(\begin{array}{l}
\left(\begin{array}{l} (1.wait \vee 2.wait) \\ \wedge\, ref' \subseteq \left(\begin{array}{l} ((1.ref \cup 2.ref) \cap cs) \\ \cup((1.ref \cap 2.ref) \setminus cs) \end{array} \right) \end{array} \right) \\
\lhd wait' \rhd \\
(\neg\, 1.wait \wedge \neg\, 2.wait \wedge MSt)
\end{array} \right)
\end{array} \right)
\end{array} \right)
\end{array} \right\} \\
\cup \left\{ \begin{array}{l}
(tr' - tr, ref' \cup \{\checkmark\}) \\
\mid okay \wedge wait' \wedge \\
\quad \left(\begin{array}{l}
\left(\begin{array}{l} (P^t_f;\ U1(out\alpha\, P)) \\ \wedge\, (Q^t_f;\ U2(out\alpha\, Q)) \end{array} \right)_{+\{v,tr\}} ; \\
\left(\begin{array}{l}
tr' - tr \in (1.tr - tr \parallel_{cs} 2.tr - tr) \\
\wedge\, 1.tr \upharpoonright cs = 2.tr \upharpoonright cs \\
\wedge \left(\begin{array}{l}
\left(\begin{array}{l} (1.wait \vee 2.wait) \\ \wedge\, ref' \subseteq \left(\begin{array}{l} ((1.ref \cup 2.ref) \cap cs) \\ \cup((1.ref \cap 2.ref) \setminus cs) \end{array} \right) \end{array} \right) \\
\lhd wait' \rhd \\
(\neg\, 1.wait \wedge \neg\, 2.wait \wedge MSt)
\end{array} \right)
\end{array} \right)
\end{array} \right)
\end{array} \right\} \\
\cup \left\{ \begin{array}{l}
((tr' - tr) \frown \langle \checkmark \rangle, ref') \\
\mid okay \wedge \neg\, wait' \wedge \\
\quad \left(\begin{array}{l}
\left(\begin{array}{l} (P^t_f;\ U1(out\alpha\, P)) \\ \wedge\, (Q^t_f;\ U2(out\alpha\, Q)) \end{array} \right)_{+\{v,tr\}} ; \\
\left(\begin{array}{l}
tr' - tr \in (1.tr - tr \parallel_{cs} 2.tr - tr) \\
\wedge\, 1.tr \upharpoonright cs = 2.tr \upharpoonright cs \\
\wedge \left(\begin{array}{l}
\left(\begin{array}{l} (1.wait \vee 2.wait) \\ \wedge\, ref' \subseteq \left(\begin{array}{l} ((1.ref \cup 2.ref) \cap cs) \\ \cup((1.ref \cap 2.ref) \setminus cs) \end{array} \right) \end{array} \right) \\
\lhd wait' \rhd \\
(\neg\, 1.wait \wedge \neg\, 2.wait \wedge MSt)
\end{array} \right)
\end{array} \right)
\end{array} \right)
\end{array} \right\} \\
\cup \left\{ \begin{array}{l}
((tr' - tr) \frown \langle \checkmark \rangle, ref' \cup \{\checkmark\}) \mid \\
okay \wedge \neg\, wait' \wedge \\
\quad \left(\begin{array}{l}
\left(\begin{array}{l} (P^t_f;\ U1(out\alpha\, P)) \\ \wedge\, (Q^t_f;\ U2(out\alpha\, Q)) \end{array} \right)_{+\{v,tr\}} ; \\
\left(\begin{array}{l}
tr' - tr \in (1.tr - tr \parallel_{cs} 2.tr - tr) \\
\wedge\, 1.tr \upharpoonright cs = 2.tr \upharpoonright cs \\
\wedge \left(\begin{array}{l}
\left(\begin{array}{l} (1.wait \vee 2.wait) \\ \wedge\, ref' \subseteq \left(\begin{array}{l} ((1.ref \cup 2.ref) \cap cs) \\ \cup((1.ref \cap 2.ref) \setminus cs) \end{array} \right) \end{array} \right) \\
\lhd wait' \rhd \\
(\neg\, 1.wait \wedge \neg\, 2.wait \wedge MSt)
\end{array} \right)
\end{array} \right)
\end{array} \right)
\end{array} \right\}
\end{array}
$$

[Sequence, PC and ST]

$$
= \left\{ \begin{array}{l} (tr' - tr, ref') \\ \mid okay \wedge wait' \wedge \\ \qquad \left(\begin{array}{l} \left(\begin{array}{l} (P_f^t;\ U1(out\alpha\, P)) \\ \wedge\ (Q_f^t;\ U2(out\alpha\, Q)) \end{array} \right)_{+\{v,tr\}} ; \\ \left(\begin{array}{l} tr' - tr \in (1.tr - tr \parallel_{cs} 2.tr - tr) \\ \wedge\ 1.tr \upharpoonright cs = 2.tr \upharpoonright cs \\ \wedge \left(\begin{array}{l} (1.wait \vee 2.wait) \\ \wedge\ ref' \subseteq \left(\begin{array}{l} ((1.ref \cup 2.ref) \cap cs) \\ \cup((1.ref \cap 2.ref) \setminus cs) \end{array} \right) \end{array} \right) \end{array} \right) \end{array} \right) \end{array} \right\}
$$

$$
\cup \left\{ \begin{array}{l} (tr' - tr, ref') \\ \mid okay \wedge \neg\ wait' \wedge \\ \qquad \left(\begin{array}{l} \left(\begin{array}{l} (P_f^t;\ U1(out\alpha\, P)) \\ \wedge\ (Q_f^t;\ U2(out\alpha\, Q)) \end{array} \right)_{+\{v,tr\}} ; \\ \left(\begin{array}{l} tr' - tr \in (1.tr - tr \parallel_{cs} 2.tr - tr) \\ \wedge\ 1.tr \upharpoonright cs = 2.tr \upharpoonright cs \\ \wedge \neg\ 1.wait \wedge \neg\ 2.wait \wedge MSt \end{array} \right) \end{array} \right) \end{array} \right\}
$$

$$
\cup \left\{ \begin{array}{l} (tr' - tr, ref' \cup \{\checkmark\}) \\ \mid okay \wedge wait' \wedge \\ \qquad \left(\begin{array}{l} \left(\begin{array}{l} (P_f^t;\ U1(out\alpha\, P)) \\ \wedge\ (Q_f^t;\ U2(out\alpha\, Q)) \end{array} \right)_{+\{v,tr\}} ; \\ \left(\begin{array}{l} tr' - tr \in (1.tr - tr \parallel_{cs} 2.tr - tr) \\ \wedge\ 1.tr \upharpoonright cs = 2.tr \upharpoonright cs \\ \wedge \left(\begin{array}{l} (1.wait \vee 2.wait) \\ \wedge\ ref' \subseteq \left(\begin{array}{l} ((1.ref \cup 2.ref) \cap cs) \\ \cup((1.ref \cap 2.ref) \setminus cs) \end{array} \right) \end{array} \right) \end{array} \right) \end{array} \right) \end{array} \right\}
$$

$$
\cup \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref') \\ \mid okay \wedge \neg\ wait' \wedge \\ \qquad \left(\begin{array}{l} \left(\begin{array}{l} (P_f^t;\ U1(out\alpha\, P)) \\ \wedge\ (Q_f^t;\ U2(out\alpha\, Q)) \end{array} \right)_{+\{v,tr\}} ; \\ \left(\begin{array}{l} tr' - tr \in (1.tr - tr \parallel_{cs} 2.tr - tr) \\ \wedge\ 1.tr \upharpoonright cs = 2.tr \upharpoonright cs \\ \neg\ 1.wait \wedge \neg\ 2.wait \wedge MSt \end{array} \right) \end{array} \right) \end{array} \right\}
$$

$$
\cup \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref' \cup \{\checkmark\}) \mid \\ okay \wedge \neg\ wait' \wedge \\ \qquad \left(\begin{array}{l} \left(\begin{array}{l} (P_f^t;\ U1(out\alpha\, P)) \\ \wedge\ (Q_f^t;\ U2(out\alpha\, Q)) \end{array} \right)_{+\{v,tr\}} ; \\ \left(\begin{array}{l} tr' - tr \in (1.tr - tr \parallel_{cs} 2.tr - tr) \\ \wedge\ 1.tr \upharpoonright cs = 2.tr \upharpoonright cs \\ \wedge \neg\ 1.wait \wedge \neg\ 2.wait \wedge MSt \end{array} \right) \end{array} \right) \end{array} \right\}
$$

At this point of the proof, for conciseness, we need to introduce further notation to give names to predicates.

- Separate execution of P and Q

$$PQ \mathrel{\widehat{=}} \left(\begin{array}{l} (P_f^t;\ U1(out\alpha\, P)) \\ \wedge\ (Q_f^t;\ U2(out\alpha\, Q)) \end{array}\right)_{+\{v,tr\}}$$

- Merge for P and Q not waiting

$$M_{P_f Q_f} \mathrel{\widehat{=}} \left(\begin{array}{l} tr' - tr \in (1.tr - tr \parallel_{cs} 2.tr - tr) \\ \wedge\ 1.tr \upharpoonright cs = 2.tr \upharpoonright cs \\ \wedge\ \neg\ 1.wait \wedge \neg\ 2.wait \wedge MSt \end{array}\right)$$

- Merge for P and Q waiting

$$M_{P_t Q_t} \mathrel{\widehat{=}} \left(\begin{array}{l} tr' - tr \in (1.tr - tr \parallel_{cs} 2.tr - tr) \\ \wedge\ 1.tr \upharpoonright cs = 2.tr \upharpoonright cs \\ \wedge\ 1.wait \wedge 2.wait \\ \wedge\ ref' \subseteq \left(\begin{array}{l} ((1.ref \cup 2.ref) \cap cs) \\ \cup((1.ref \cap 2.ref) \setminus cs) \end{array}\right) \end{array}\right)$$

- Merge for P waiting and Q not waiting

$$M_{P_t Q_f} \mathrel{\widehat{=}} \left(\begin{array}{l} tr' - tr \in (1.tr - tr \parallel_{cs} 2.tr - tr) \\ \wedge\ 1.tr \upharpoonright cs = 2.tr \upharpoonright cs \\ \wedge\ 1.wait \wedge \neg\ 2.wait \\ \wedge\ ref' \subseteq \left(\begin{array}{l} ((1.ref \cup 2.ref) \cap cs) \\ \cup((1.ref \cap 2.ref) \setminus cs) \end{array}\right) \end{array}\right)$$

- Merge for P not waiting and Q waiting

$$M_{P_f Q_t} \mathrel{\widehat{=}} \left(\begin{array}{l} tr' - tr \in (1.tr - tr \parallel_{cs} 2.tr - tr) \\ \wedge\ 1.tr \upharpoonright cs = 2.tr \upharpoonright cs \\ \wedge\ \neg\ 1.wait \wedge 2.wait \\ \wedge\ ref' \subseteq \left(\begin{array}{l} ((1.ref \cup 2.ref) \cap cs) \\ \cup((1.ref \cap 2.ref) \setminus cs) \end{array}\right) \end{array}\right)$$

[Sequence, PC and ST]

$= $ `(A)`
$\{(tr' - tr, ref') \mid okay \wedge wait' \wedge (PQ;\ M_{P_t Q_t})\}$
`(B)`
$\cup \{(tr' - tr, ref') \mid okay \wedge wait' \wedge (PQ;\ M_{P_t Q_f})\}$
`(C)`
$\cup \{(tr' - tr, ref') \mid okay \wedge wait' \wedge (PQ;\ M_{P_f Q_t})\}$
`(D)`
$\cup \{(tr' - tr, ref') \mid okay \wedge \neg\ wait' \wedge (PQ;\ M_{P_f Q_f})\}$
`(E)`
$\cup \{(tr' - tr, ref' \cup \{\checkmark\}) \mid okay \wedge wait' \wedge (PQ;\ M_{P_t Q_t})\}$
`(F)`
$\cup \{(tr' - tr, ref' \cup \{\checkmark\}) \mid okay \wedge wait' \wedge (PQ;\ M_{P_t Q_f})\}$
`(G)`
$\cup \{(tr' - tr, ref' \cup \{\checkmark\}) \mid okay \wedge wait' \wedge (PQ;\ M_{P_f Q_t})\}$
`(H)`
$\cup \{((tr' - tr) \frown \langle\checkmark\rangle, ref') \mid okay \wedge \neg\ wait' \wedge (PQ;\ M_{P_f Q_f})\}$
`(I)`
$\cup \{((tr' - tr) \frown \langle\checkmark\rangle, ref' \cup \{\checkmark\}) \mid okay \wedge \neg\ wait' \wedge (PQ;\ M_{P_f Q_f})\}$

At this step, we have a one-to-one correspondence. They are:

- `(A)` is equivalent to `(1.1)` by Lemma J.37
- `(B)` is equivalent to `(1.2)` by Lemma J.38
- `(C)` is equivalent to `(1.3)` by Lemma J.39
- `(D)` is equivalent to `(1.4)` by Lemma J.40
- `(E)` is equivalent to `(2.1)` by Lemma J.41
- `(F)` is equivalent to `(5.2)` by Lemma J.42
- `(G)` is equivalent to `(2.2)` by Lemma J.43
- `(H)` is equivalent to `(11)` by Lemma J.44
- `(I)` is equivalent to `(12)` by Lemma J.45

Applying these lemmas, we continue the proof below.

$=$

`(1.1)`

$$\left\{\begin{array}{l}(u, Y \cup Z) \\ \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \quad \wedge \exists\, s, t \bullet (s, Y) \in \{(tr' - tr, ref') \mid okay \wedge wait' \wedge (P)^t_f\} \\ \qquad\qquad\quad \wedge (t, Z) \in \{(tr' - tr, ref') \mid okay \wedge wait' \wedge (Q)^t_f\} \\ \qquad\qquad\quad \wedge u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t\end{array}\right\}$$

(1.2)

$$\cup \left\{\begin{array}{l}(u, Y \cup Z) \\ \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \quad \wedge \exists\, s, t \bullet (s, Y) \in \{(tr' - tr, ref') \mid okay \wedge wait' \wedge (P)^t_f\} \\ \qquad\qquad\quad \wedge (t, Z) \in \{(tr' - tr, ref') \mid okay \wedge \neg\, wait' \wedge (Q)^t_f\} \\ \qquad\qquad\quad \wedge u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t\end{array}\right\}$$

(1.3)

$$\cup \left\{\begin{array}{l}(u, Y \cup Z) \\ \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \quad \wedge \exists\, s, t \bullet (s, Y) \in \{(tr' - tr, ref') \mid okay \wedge \neg\, wait' \wedge (P)^t_f\} \\ \qquad\qquad\quad \wedge (t, Z) \in \{(tr' - tr, ref') \mid okay \wedge wait' \wedge (Q)^t_f\} \\ \qquad\qquad\quad \wedge u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t\end{array}\right\}$$

(1.4)

$$\cup \left\{\begin{array}{l}(u, Y \cup Z) \\ \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \quad \wedge \exists\, s, t \bullet (s, Y) \in \{(tr' - tr, ref') \mid okay \wedge \neg\, wait' \wedge (P)^t_f\} \\ \qquad\qquad\quad \wedge (t, Z) \in \{(tr' - tr, ref') \mid okay \wedge \neg\, wait' \wedge (Q)^t_f\} \\ \qquad\qquad\quad \wedge u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t\end{array}\right\}$$

(2.1)

$$\cup \left\{\begin{array}{l}(u, Y \cup Z \cup \{\checkmark\}) \\ \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \quad \wedge \exists\, s, t \bullet (s, Y) \in \{(tr' - tr, ref') \mid okay \wedge wait' \wedge (P)^t_f\} \\ \qquad\qquad\quad \wedge (t, Z) \in \{(tr' - tr, ref') \mid okay \wedge wait' \wedge (Q)^t_f\} \\ \qquad\qquad\quad \wedge u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t\end{array}\right\}$$

(2.2)

$$\cup \left\{ \begin{array}{l} (u,\, Y \cup Z \cup \{\checkmark\}) \\ \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \quad \wedge \exists\, s, t \bullet (s,\, Y) \in \{(tr' - tr, ref') \mid okay \wedge \neg\, wait' \wedge (P)^t_f\} \\ \qquad\qquad \wedge\, (t, Z) \in \{(tr' - tr, ref') \mid okay \wedge wait' \wedge (Q)^t_f\} \\ \qquad\qquad \wedge\, u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array} \right\}$$

(5.2)

$$\cup \left\{ \begin{array}{l} (u,\, Y \cup Z \cup \{\checkmark\}) \\ \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \quad \wedge \exists\, s, t \bullet (s,\, Y) \in \{(tr' - tr, ref') \mid okay \wedge wait' \wedge (P)^t_f\} \\ \qquad\qquad \wedge\, (t, Z) \in \{(tr' - tr, ref') \mid okay \wedge \neg\, wait' \wedge (Q)^t_f\} \\ \qquad\qquad \wedge\, u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array} \right\}$$

(11)

$$\cup \left\{ \begin{array}{l} (u \frown \langle\checkmark\rangle,\, Y \cup Z) \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \qquad\qquad\qquad \wedge \exists\, s, t \bullet (s,\, Y) \in \{((tr' - tr), ref') \mid okay \wedge \neg\, wait' \wedge (P)^t_f\} \\ \qquad\qquad\qquad\qquad \wedge\, (t, Z) \in \{((tr' - tr), ref') \mid okay \wedge \neg\, wait' \wedge (Q)^t_f\} \\ \qquad\qquad\qquad\qquad \wedge\, u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array} \right\}$$

(12)

$$\cup \left\{ \begin{array}{l} (u \frown \langle\checkmark\rangle,\, Y \cup Z \cup \{\checkmark\}) \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \qquad\qquad\qquad \wedge \exists\, s, t \bullet (s,\, Y) \in \{((tr' - tr), ref') \mid okay \wedge \neg\, wait' \wedge (P)^t_f \\ \qquad\qquad\qquad\qquad \wedge\, (t, Z) \in \left\{ \begin{array}{l} ((tr' - tr), ref') \\ \mid okay \wedge \neg\, wait' \wedge (Q)^t_f \end{array} \right\} \\ \qquad\qquad\qquad\qquad \wedge\, u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array} \right.$$

At this step, we use set theory to repeat some elements. The repetitions are:

- Repeating (2.1): (5.1) and (6)
- Repeating (12): (15) and (16)

$=$

(1.1)

$$\left\{ \begin{array}{l} (u,\, Y \cup Z) \\ \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \quad \wedge \exists\, s, t \bullet (s,\, Y) \in \{(tr' - tr, ref') \mid okay \wedge wait' \wedge (P)^t_f\} \\ \qquad\qquad \wedge\, (t, Z) \in \{(tr' - tr, ref') \mid okay \wedge wait' \wedge (Q)^t_f\} \\ \qquad\qquad \wedge\, u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array} \right\}$$

(1.2)

$$\cup \left\{ \begin{array}{l} (u, Y \cup Z) \\ \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \quad \wedge \exists\, s, t \bullet (s, Y) \in \{(tr' - tr, ref') \mid okay \wedge wait' \wedge (P)^t_f\} \\ \qquad\qquad \wedge (t, Z) \in \{(tr' - tr, ref') \mid okay \wedge \neg\, wait' \wedge (Q)^t_f\} \\ \qquad\qquad \wedge u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array} \right\}$$

(1.3)

$$\cup \left\{ \begin{array}{l} (u, Y \cup Z) \\ \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \quad \wedge \exists\, s, t \bullet (s, Y) \in \{(tr' - tr, ref') \mid okay \wedge \neg\, wait' \wedge (P)^t_f\} \\ \qquad\qquad \wedge (t, Z) \in \{(tr' - tr, ref') \mid okay \wedge wait' \wedge (Q)^t_f\} \\ \qquad\qquad \wedge u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array} \right\}$$

(1.4)

$$\cup \left\{ \begin{array}{l} (u, Y \cup Z) \\ \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \quad \wedge \exists\, s, t \bullet (s, Y) \in \{(tr' - tr, ref') \mid okay \wedge \neg\, wait' \wedge (P)^t_f\} \\ \qquad\qquad \wedge (t, Z) \in \{(tr' - tr, ref') \mid okay \wedge \neg\, wait' \wedge (Q)^t_f\} \\ \qquad\qquad \wedge u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array} \right\}$$

(2.1)

$$\cup \left\{ \begin{array}{l} (u, Y \cup Z \cup \{\checkmark\}) \\ \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \quad \wedge \exists\, s, t \bullet (s, Y) \in \{(tr' - tr, ref') \mid okay \wedge wait' \wedge (P)^t_f\} \\ \qquad\qquad \wedge (t, Z) \in \{(tr' - tr, ref') \mid okay \wedge wait' \wedge (Q)^t_f\} \\ \qquad\qquad \wedge u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array} \right\}$$

(2.2)

$$\cup \left\{ \begin{array}{l} (u, Y \cup Z \cup \{\checkmark\}) \\ \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \quad \wedge \exists\, s, t \bullet (s, Y) \in \{(tr' - tr, ref') \mid okay \wedge \neg\, wait' \wedge (P)^t_f\} \\ \qquad\qquad \wedge (t, Z) \in \{(tr' - tr, ref') \mid okay \wedge wait' \wedge (Q)^t_f\} \\ \qquad\qquad \wedge u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array} \right\}$$

(5.1)

$$\cup \left\{ \begin{array}{l} (u, Y \cup Z \cup \{\checkmark\}) \\ \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \quad \wedge \exists\, s, t \bullet (s, Y) \in \{(tr' - tr, ref') \mid okay \wedge wait' \wedge (P)^t_f\} \\ \qquad\qquad \wedge (t, Z) \in \{(tr' - tr, ref') \mid okay \wedge wait' \wedge (Q)^t_f\} \\ \qquad\qquad \wedge u \in s \underset{\mathsf{cs}^{\checkmark}}{\|} t \end{array} \right\}$$

(5.2)

$$\cup \left\{ \begin{array}{l} (u, Y \cup Z \cup \{\checkmark\}) \\ \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \quad \wedge \exists\, s, t \bullet (s, Y) \in \{(tr' - tr, ref') \mid okay \wedge wait' \wedge (P)^t_f\} \\ \qquad\qquad \wedge (t, Z) \in \{(tr' - tr, ref') \mid okay \wedge \neg\, wait' \wedge (Q)^t_f\} \\ \qquad\qquad \wedge u \in s \underset{\mathsf{cs}^{\checkmark}}{\|} t \end{array} \right\}$$

(6)

$$\cup \left\{ \begin{array}{l} (u, Y \cup Z \cup \{\checkmark\}) \\ \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \quad \wedge \exists\, s, t \bullet (s, Y) \in \{(tr' - tr, ref') \mid okay \wedge wait' \wedge (P)^t_f\} \\ \qquad\qquad \wedge (t, Z) \in \{(tr' - tr, ref') \mid okay \wedge wait' \wedge (Q)^t_f\} \\ \qquad\qquad \wedge u \in s \underset{\mathsf{cs}^{\checkmark}}{\|} t \end{array} \right\}$$

(11)

$$\cup \left\{ \begin{array}{l} (u \frown \langle\checkmark\rangle, Y \cup Z) \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \qquad\qquad\qquad\qquad \wedge \exists\, s, t \bullet (s, Y) \in \{((tr' - tr), ref') \mid okay \wedge \neg\, wait' \wedge (P)^t_f\} \\ \qquad\qquad\qquad\qquad\qquad\qquad \wedge (t, Z) \in \{((tr' - tr), ref') \mid okay \wedge \neg\, wait' \wedge (Q)^t_f\} \\ \qquad\qquad\qquad\qquad\qquad\qquad \wedge u \in s \underset{\mathsf{cs}^{\checkmark}}{\|} t \end{array} \right\}$$

(12)

$$\cup \left\{ \begin{array}{l} (u \frown \langle\checkmark\rangle, Y \cup Z \cup \{\checkmark\}) \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \qquad\qquad\qquad\qquad\qquad \wedge \exists\, s, t \bullet (s, Y) \in \{((tr' - tr), ref') \mid okay \wedge \neg\, wait' \wedge (P)^t_f \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad \wedge (t, Z) \in \left\{ \begin{array}{l} ((tr' - tr), ref') \\ \mid okay \wedge \neg\, wait' \wedge (Q)^t_f \end{array} \right\} \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad \wedge u \in s \underset{\mathsf{cs}^{\checkmark}}{\|} t \end{array} \right.$$

(15)

$$\cup \left\{ \begin{array}{l} (u \frown \langle \checkmark \rangle, Y \cup Z \cup \{\checkmark\}) \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \quad \wedge \exists\, s, t \bullet (s, Y) \in \left\{ \begin{array}{l} ((tr' - tr), ref') \\ \mid okay \wedge \neg\, wait' \wedge (P)^t_f \end{array} \right\} \\ \quad \wedge (t, Z) \in \left\{ \begin{array}{l} ((tr' - tr), ref') \\ \mid okay \wedge \neg\, wait' \wedge (Q)^t_f \end{array} \right\} \\ \quad \wedge u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array} \right\}$$

(16)

$$\cup \left\{ \begin{array}{l} (u \frown \langle \checkmark \rangle, Y \cup Z \cup \{\checkmark\}) \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \quad \wedge \exists\, s, t \bullet (s, Y) \in \left\{ \begin{array}{l} ((tr' - tr), ref') \\ \mid okay \wedge \neg\, wait' \wedge (P)^t_f \end{array} \right\} \\ \quad \wedge (t, Z) \in \left\{ \begin{array}{l} ((tr' - tr), ref') \\ \mid okay \wedge \neg\, wait' \wedge (Q)^t_f \end{array} \right\} \\ \quad \wedge u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array} \right\}$$

[Lemma J.49, SC, ST]

$=$

(1.1)

$$\left\{ \begin{array}{l} (u, Y \cup Z) \\ \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \quad \wedge \exists\, s, t \bullet (s, Y) \in \{(tr' - tr, ref') \mid okay \wedge wait' \wedge (P)^t_f\} \\ \qquad \wedge (t, Z) \in \{(tr' - tr, ref') \mid okay \wedge wait' \wedge (Q)^t_f\} \\ \qquad \wedge u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array} \right\}$$

(1.2)

$$\cup \left\{ \begin{array}{l} (u, Y \cup Z) \\ \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \quad \wedge \exists\, s, t \bullet (s, Y) \in \{(tr' - tr, ref') \mid okay \wedge wait' \wedge (P)^t_f\} \\ \qquad \wedge (t, Z) \in \{(tr' - tr, ref') \mid okay \wedge \neg\, wait' \wedge (Q)^t_f\} \\ \qquad \wedge u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array} \right\}$$

(1.3)

$$\cup \left\{ \begin{array}{l} (u, Y \cup Z) \\ \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \quad \wedge \exists\, s, t \bullet (s, Y) \in \{(tr' - tr, ref') \mid okay \wedge \neg\, wait' \wedge (P)^t_f\} \\ \qquad \wedge (t, Z) \in \{(tr' - tr, ref') \mid okay \wedge wait' \wedge (Q)^t_f\} \\ \qquad \wedge u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array} \right\}$$

(1.4)

$$\cup \left\{ \begin{array}{l} (u, Y \cup Z) \\ \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \quad \wedge \exists\, s, t \bullet (s, Y) \in \{(tr' - tr, ref') \mid okay \wedge \neg\, wait' \wedge (P)^t_f\} \\ \qquad\qquad \wedge (t, Z) \in \{(tr' - tr, ref') \mid okay \wedge \neg\, wait' \wedge (Q)^t_f\} \\ \qquad\qquad \wedge u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array} \right\}$$

(2.1)

$$\cup \left\{ \begin{array}{l} (u, Y \cup Z \cup \{\checkmark\}) \\ \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \quad \wedge \exists\, s, t \bullet (s, Y) \in \{(tr' - tr, ref') \mid okay \wedge wait' \wedge (P)^t_f\} \\ \qquad\qquad \wedge (t, Z) \in \{(tr' - tr, ref') \mid okay \wedge wait' \wedge (Q)^t_f\} \\ \qquad\qquad \wedge u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array} \right\}$$

(2.2)

$$\cup \left\{ \begin{array}{l} (u, Y \cup Z \cup \{\checkmark\}) \\ \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \quad \wedge \exists\, s, t \bullet (s, Y) \in \{(tr' - tr, ref') \mid okay \wedge \neg\, wait' \wedge (P)^t_f\} \\ \qquad\qquad \wedge (t, Z) \in \{(tr' - tr, ref') \mid okay \wedge wait' \wedge (Q)^t_f\} \\ \qquad\qquad \wedge u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array} \right\}$$

(5.1)

$$\cup \left\{ \begin{array}{l} (u, Y \cup Z \cup \{\checkmark\}) \\ \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \quad \wedge \exists\, s, t \bullet (s, Y) \in \{(tr' - tr, ref') \mid okay \wedge wait' \wedge (P)^t_f\} \\ \qquad\qquad \wedge (t, Z) \in \{(tr' - tr, ref') \mid okay \wedge wait' \wedge (Q)^t_f\} \\ \qquad\qquad \wedge u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array} \right\}$$

(5.2)

$$\cup \left\{ \begin{array}{l} (u, Y \cup Z \cup \{\checkmark\}) \\ \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \quad \wedge \exists\, s, t \bullet (s, Y) \in \{(tr' - tr, ref') \mid okay \wedge wait' \wedge (P)^t_f\} \\ \qquad\qquad \wedge (t, Z) \in \{(tr' - tr, ref') \mid okay \wedge \neg\, wait' \wedge (Q)^t_f\} \\ \qquad\qquad \wedge u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array} \right\}$$

(6)

$$\cup \left\{ \begin{array}{l} (u, Y \cup Z \cup \{\checkmark\}) \\ \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \quad \wedge \exists\, s, t \bullet (s, Y) \in \{(tr' - tr, ref') \mid okay \wedge wait' \wedge (P)^t_f\} \\ \qquad\qquad\quad \wedge (t, Z) \in \{(tr' - tr, ref') \mid okay \wedge wait' \wedge (Q)^t_f\} \\ \qquad\qquad\quad \wedge u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array} \right\}$$

(11)

$$\cup \left\{ \begin{array}{l} (u, Y \cup Z) \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \qquad\qquad\qquad \wedge \exists\, s, t \bullet (s, Y) \in \{((tr' - tr) \frown \langle \checkmark \rangle, ref') \mid okay \wedge \neg\, wait' \wedge (P)^t_f\} \\ \qquad\qquad\qquad\qquad\qquad \wedge (t, Z) \in \{((tr' - tr) \frown \langle \checkmark \rangle, ref') \mid okay \wedge \neg\, wait' \wedge (Q)^t_f\} \\ \qquad\qquad\qquad\qquad\qquad \wedge u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array} \right\}$$

(12)

$$\cup \left\{ \begin{array}{l} (u, Y \cup Z \cup \{\checkmark\}) \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \qquad\qquad\qquad \wedge \exists\, s, t \bullet (s, Y) \in \{((tr' - tr) \frown \langle \checkmark \rangle, ref') \mid okay \wedge \neg\, wait' \wedge (P)^t_f \\ \qquad\qquad\qquad\qquad\qquad \wedge (t, Z) \in \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref') \\ \mid okay \wedge \neg\, wait' \wedge (Q)^t_f \end{array} \right\} \\ \qquad\qquad\qquad\qquad\qquad \wedge u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array} \right.$$

(15)

$$\cup \left\{ \begin{array}{l} (u, Y \cup Z \cup \{\checkmark\}) \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \qquad\qquad\qquad \wedge \exists\, s, t \bullet (s, Y) \in \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref') \\ \mid okay \wedge \neg\, wait' \wedge (P)^t_f \end{array} \right\} \\ \qquad\qquad\qquad\qquad\qquad \wedge (t, Z) \in \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref') \\ \mid okay \wedge \neg\, wait' \wedge (Q)^t_f \end{array} \right\} \\ \qquad\qquad\qquad\qquad\qquad \wedge u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array} \right\}$$

(16)

$$\cup \left\{ \begin{array}{l} (u, Y \cup Z \cup \{\checkmark\}) \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \qquad\qquad\qquad \wedge \exists\, s, t \bullet (s, Y) \in \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref') \\ \mid okay \wedge \neg\, wait' \wedge (P)^t_f \end{array} \right\} \\ \qquad\qquad\qquad\qquad\qquad \wedge (t, Z) \in \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref') \\ \mid okay \wedge \neg\, wait' \wedge (Q)^t_f \end{array} \right\} \\ \qquad\qquad\qquad\qquad\qquad \wedge u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array} \right\}$$

[ST]

$=$

(1.1)

$$\left\{\begin{array}{l}(u, Y \cup Z)\\ \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\})\\ \quad \wedge \exists\, s, t \bullet (s, Y) \in \{(tr' - tr, ref') \mid okay \wedge wait' \wedge (P)^t_f\}\\ \qquad\qquad \wedge (t, Z) \in \{(tr' - tr, ref') \mid okay \wedge wait' \wedge (Q)^t_f\}\\ \qquad\qquad \wedge u \in s \underset{\mathsf{cs}^\checkmark}{\parallel} t\end{array}\right\}$$

(1.2)

$$\cup \left\{\begin{array}{l}(u, Y \cup Z)\\ \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\})\\ \quad \wedge \exists\, s, t \bullet (s, Y) \in \{(tr' - tr, ref') \mid okay \wedge wait' \wedge (P)^t_f\}\\ \qquad\qquad \wedge (t, Z) \in \{(tr' - tr, ref') \mid okay \wedge \neg\, wait' \wedge (Q)^t_f\}\\ \qquad\qquad \wedge u \in s \underset{\mathsf{cs}^\checkmark}{\parallel} t\end{array}\right\}$$

(1.3)

$$\cup \left\{\begin{array}{l}(u, Y \cup Z)\\ \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\})\\ \quad \wedge \exists\, s, t \bullet (s, Y) \in \{(tr' - tr, ref') \mid okay \wedge \neg\, wait' \wedge (P)^t_f\}\\ \qquad\qquad \wedge (t, Z) \in \{(tr' - tr, ref') \mid okay \wedge wait' \wedge (Q)^t_f\}\\ \qquad\qquad \wedge u \in s \underset{\mathsf{cs}^\checkmark}{\parallel} t\end{array}\right\}$$

(1.4)

$$\cup \left\{\begin{array}{l}(u, Y \cup Z)\\ \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\})\\ \quad \wedge \exists\, s, t \bullet (s, Y) \in \{(tr' - tr, ref') \mid okay \wedge \neg\, wait' \wedge (P)^t_f\}\\ \qquad\qquad \wedge (t, Z) \in \{(tr' - tr, ref') \mid okay \wedge \neg\, wait' \wedge (Q)^t_f\}\\ \qquad\qquad \wedge u \in s \underset{\mathsf{cs}^\checkmark}{\parallel} t\end{array}\right\}$$

(2.1)

$$\cup \left\{\begin{array}{l}(u, Y \cup Z \cup \{\checkmark\})\\ \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\})\\ \quad \wedge \exists\, s, t \bullet (s, Y) \in \{(tr' - tr, ref') \mid okay \wedge wait' \wedge (P)^t_f\}\\ \qquad\qquad \wedge (t, Z) \in \{(tr' - tr, ref') \mid okay \wedge wait' \wedge (Q)^t_f\}\\ \qquad\qquad \wedge u \in s \underset{\mathsf{cs}^\checkmark}{\parallel} t\end{array}\right\}$$

(2.2)

$$\cup \left\{ \begin{array}{l} (u, Y \cup Z \cup \{\checkmark\}) \\ \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \quad \wedge \exists s, t \bullet (s, Y) \in \{(tr' - tr, ref') \mid okay \wedge \neg\, wait' \wedge (P)^t_f\} \\ \qquad\qquad \wedge (t, Z) \in \{(tr' - tr, ref') \mid okay \wedge wait' \wedge (Q)^t_f\} \\ \qquad\qquad \wedge u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array} \right\}$$

(3.1)

$\cup \{\}$

(3.2)

$\cup \{\}$

(4.1)

$\cup \{\}$

(4.2)

$\cup \{\}$

(5.1)

$$\cup \left\{ \begin{array}{l} (u, Y \cup Z \cup \{\checkmark\}) \\ \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \quad \wedge \exists s, t \bullet (s, Y) \in \{(tr' - tr, ref') \mid okay \wedge wait' \wedge (P)^t_f\} \\ \qquad\qquad \wedge (t, Z) \in \{(tr' - tr, ref') \mid okay \wedge wait' \wedge (Q)^t_f\} \\ \qquad\qquad \wedge u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array} \right\}$$

(5.2)

$$\cup \left\{ \begin{array}{l} (u, Y \cup Z \cup \{\checkmark\}) \\ \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \quad \wedge \exists s, t \bullet (s, Y) \in \{(tr' - tr, ref') \mid okay \wedge wait' \wedge (P)^t_f\} \\ \qquad\qquad \wedge (t, Z) \in \{(tr' - tr, ref') \mid okay \wedge \neg\, wait' \wedge (Q)^t_f\} \\ \qquad\qquad \wedge u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array} \right\}$$

(6)

$$\cup \left\{ \begin{array}{l} (u, Y \cup Z \cup \{\checkmark\}) \\ \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \quad \wedge \exists s, t \bullet (s, Y) \in \{(tr' - tr, ref') \mid okay \wedge wait' \wedge (P)^t_f\} \\ \qquad\qquad \wedge (t, Z) \in \{(tr' - tr, ref') \mid okay \wedge wait' \wedge (Q)^t_f\} \\ \qquad\qquad \wedge u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array} \right\}$$

(7)

$\cup \{\}$

(8)

$\cup \{\}$

(9.1)

$\cup \{\}$

(9.2)

$\cup \{\}$

(10)

$\cup \{\}$

(11)

$$\cup \left\{ \begin{array}{l} (u, Y \cup Z) \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \qquad\qquad\quad \land \exists\, s, t \bullet (s, Y) \in \{((tr' - tr) ^\frown \langle \checkmark \rangle, ref') \mid okay \land \neg\, wait' \land (P)^t_f\} \\ \qquad\qquad\qquad\qquad \land (t, Z) \in \{((tr' - tr) ^\frown \langle \checkmark \rangle, ref') \mid okay \land \neg\, wait' \land (Q)^t_f\} \\ \qquad\qquad\qquad\qquad \land u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array} \right\}$$

(12)

$$\cup \left\{ \begin{array}{l} (u, Y \cup Z \cup \{\checkmark\}) \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \qquad\qquad\qquad\quad \land \exists\, s, t \bullet (s, Y) \in \{((tr' - tr) ^\frown \langle \checkmark \rangle, ref') \mid okay \land \neg\, wait' \land (P)^t_f \\ \qquad\qquad\qquad\qquad\qquad \land (t, Z) \in \left\{ \begin{array}{l} ((tr' - tr) ^\frown \langle \checkmark \rangle, ref') \\ \mid okay \land \neg\, wait' \land (Q)^t_f \end{array} \right\} \\ \qquad\qquad\qquad\qquad\qquad \land u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array} \right.$$

(13.1)

$\cup \{\}$

(13.2)

$\cup \{\}$

(14)

$\cup \{\}$

(15)

$$\cup \left\{ \begin{array}{l} (u, Y \cup Z \cup \{\checkmark\}) \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \qquad\qquad\qquad\quad \land \exists\, s, t \bullet (s, Y) \in \left\{ \begin{array}{l} ((tr' - tr) ^\frown \langle \checkmark \rangle, ref') \\ \mid okay \land \neg\, wait' \land (P)^t_f \end{array} \right\} \\ \qquad\qquad\qquad\qquad\qquad \land (t, Z) \in \left\{ \begin{array}{l} ((tr' - tr) ^\frown \langle \checkmark \rangle, ref') \\ \mid okay \land \neg\, wait' \land (Q)^t_f \end{array} \right\} \\ \qquad\qquad\qquad\qquad\qquad \land u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array} \right\}$$

(16)

$$\cup \left\{ \begin{array}{l} (u, Y \cup Z \cup \{\checkmark\}) \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \quad \land \exists\, s, t \bullet (s, Y) \in \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref') \\ \mid okay \land \neg\, wait' \land (P)^t_f \end{array} \right\} \\ \quad \land (t, Z) \in \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref') \\ \mid okay \land \neg\, wait' \land (Q)^t_f \end{array} \right\} \\ \quad \land u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array} \right\}$$

[Lemma J.48 (tr, tr' : seq Σ and $\checkmark \notin \Sigma$), SC and ST]
[on `(3.*)`, `(4.*)`, `(7)` to `(10)`, `(13)`, `(14)`]

$=$

`(1.1)`

$$\left\{ \begin{array}{l} (u, Y \cup Z) \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \quad \land \exists\, s, t \bullet (s, Y) \in \{(tr' - tr, ref') \mid okay \land wait' \land (P)^t_f\} \\ \quad \land (t, Z) \in \{(tr' - tr, ref') \mid okay \land wait' \land (Q)^t_f\} \\ \quad \land u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array} \right\}$$

`(1.2)`

$$\cup \left\{ \begin{array}{l} (u, Y \cup Z) \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \quad \land \exists\, s, t \bullet (s, Y) \in \{(tr' - tr, ref') \mid okay \land wait' \land (P)^t_f\} \\ \quad \land (t, Z) \in \{(tr' - tr, ref') \mid okay \land \neg\, wait' \land (Q)^t_f\} \\ \quad \land u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array} \right\}$$

`(1.3)`

$$\cup \left\{ \begin{array}{l} (u, Y \cup Z) \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \quad \land \exists\, s, t \bullet (s, Y) \in \{(tr' - tr, ref') \mid okay \land \neg\, wait' \land (P)^t_f\} \\ \quad \land (t, Z) \in \{(tr' - tr, ref') \mid okay \land wait' \land (Q)^t_f\} \\ \quad \land u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array} \right\}$$

`(1.4)`

$$\cup \left\{ \begin{array}{l} (u, Y \cup Z) \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \quad \land \exists\, s, t \bullet (s, Y) \in \{(tr' - tr, ref') \mid okay \land \neg\, wait' \land (P)^t_f\} \\ \quad \land (t, Z) \in \{(tr' - tr, ref') \mid okay \land \neg\, wait' \land (Q)^t_f\} \\ \quad \land u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array} \right\}$$

`(2.1)`

$$\cup \left\{ \begin{array}{l} (u, Y \cup Z \cup \{\checkmark\}) \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \qquad \wedge \exists\, s, t \bullet (s, Y) \in \{(tr' - tr, ref') \mid okay \wedge wait' \wedge (P)^t_f\} \\ \qquad\qquad \wedge (t, Z) \in \{(tr' - tr, ref') \mid okay \wedge wait' \wedge (Q)^t_f\} \\ \qquad\qquad \wedge u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array} \right\}$$

(2.2)

$$\cup \left\{ \begin{array}{l} (u, Y \cup Z \cup \{\checkmark\}) \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \qquad \wedge \exists\, s, t \bullet (s, Y) \in \{(tr' - tr, ref') \mid okay \wedge \neg\, wait' \wedge (P)^t_f\} \\ \qquad\qquad \wedge (t, Z) \in \{(tr' - tr, ref') \mid okay \wedge wait' \wedge (Q)^t_f\} \\ \qquad\qquad \wedge u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array} \right\}$$

(3.1)

$$\cup \left\{ \begin{array}{l} (u, Y \cup Z) \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \qquad \wedge \exists\, s, t \bullet (s, Y) \in \{(tr' - tr, ref') \mid okay \wedge wait' \wedge (P)^t_f\} \\ \qquad\qquad \wedge (t, Z) \in \{((tr' - tr) \frown \langle\checkmark\rangle, ref') \mid okay \wedge \neg\, wait' \wedge (Q)^t_f\} \\ \qquad\qquad \wedge u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array} \right\}$$

(3.2)

$$\cup \left\{ \begin{array}{l} (u, Y \cup Z) \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \qquad \wedge \exists\, s, t \bullet (s, Y) \in \{(tr' - tr, ref') \mid okay \wedge \neg\, wait' \wedge (P)^t_f\} \\ \qquad\qquad \wedge (t, Z) \in \{((tr' - tr) \frown \langle\checkmark\rangle, ref') \mid okay \wedge \neg\, wait' \wedge (Q)^t_f\} \\ \qquad\qquad \wedge u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array} \right\}$$

(4.1)

$$\cup \left\{ \begin{array}{l} (u, Y \cup Z \cup \{\checkmark\}) \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \qquad \wedge \exists\, s, t \bullet (s, Y) \in \{(tr' - tr, ref') \mid okay \wedge wait' \wedge (P)^t_f\} \\ \qquad\qquad \wedge (t, Z) \in \left\{ \begin{array}{l} ((tr' - tr) \frown \langle\checkmark\rangle, ref') \\ \mid okay \wedge \neg\, wait' \wedge (Q)^t_f \end{array} \right\} \\ \qquad\qquad \wedge u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array} \right\}$$

(4.2)

$$\cup \left\{ \begin{array}{l} (u, Y \cup Z \cup \{\checkmark\}) \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \qquad \wedge \exists\, s, t \bullet (s, Y) \in \{(tr' - tr, ref') \mid okay \wedge \neg\, wait' \wedge (P)^t_f\} \\ \qquad\qquad \wedge (t, Z) \in \left\{ \begin{array}{l} ((tr' - tr) \frown \langle\checkmark\rangle, ref') \\ \mid okay \wedge \neg\, wait' \wedge (Q)^t_f \end{array} \right\} \\ \qquad\qquad \wedge u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array} \right\}$$

(5.1)

$$\cup \left\{ \begin{array}{l} (u, Y \cup Z \cup \{\checkmark\}) \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \qquad \land \exists\, s, t \bullet (s, Y) \in \{(tr' - tr, ref') \mid okay \land wait' \land (P)^t_f\} \\ \qquad\qquad \land (t, Z) \in \{(tr' - tr, ref') \mid okay \land wait' \land (Q)^t_f\} \\ \qquad\qquad \land u \in s \underset{\mathsf{cs}^\checkmark}{\parallel} t \end{array} \right\}$$

(5.2)

$$\cup \left\{ \begin{array}{l} (u, Y \cup Z \cup \{\checkmark\}) \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \qquad \land \exists\, s, t \bullet (s, Y) \in \{(tr' - tr, ref') \mid okay \land wait' \land (P)^t_f\} \\ \qquad\qquad \land (t, Z) \in \{(tr' - tr, ref') \mid okay \land \neg\, wait' \land (Q)^t_f\} \\ \qquad\qquad \land u \in s \underset{\mathsf{cs}^\checkmark}{\parallel} t \end{array} \right\}$$

(6)

$$\cup \left\{ \begin{array}{l} (u, Y \cup Z \cup \{\checkmark\}) \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \qquad \land \exists\, s, t \bullet (s, Y) \in \{(tr' - tr, ref') \mid okay \land wait' \land (P)^t_f\} \\ \qquad\qquad \land (t, Z) \in \{(tr' - tr, ref') \mid okay \land wait' \land (Q)^t_f\} \\ \qquad\qquad \land u \in s \underset{\mathsf{cs}^\checkmark}{\parallel} t \end{array} \right\}$$

(7)

$$\cup \left\{ \begin{array}{l} (u, Y \cup Z \cup \{\checkmark\}) \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \qquad \land \exists\, s, t \bullet (s, Y) \in \{(tr' - tr, ref') \mid okay \land wait' \land (P)^t_f\} \\ \qquad\qquad \land (t, Z) \in \{((tr' - tr) \frown \langle \checkmark \rangle, ref') \mid okay \land \neg\, wait' \land (Q \\ \qquad\qquad \land u \in s \underset{\mathsf{cs}^\checkmark}{\parallel} t \end{array} \right.$$

(8)

$$\cup \left\{ \begin{array}{l} (u, Y \cup Z \cup \{\checkmark\}) \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \qquad \land \exists\, s, t \bullet (s, Y) \in \{(tr' - tr, ref') \mid okay \land wait' \land (P)^t_f\} \\ \qquad\qquad \land (t, Z) \in \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref') \\ \mid okay \land \neg\, wait' \land (Q)^t_f \end{array} \right\} \\ \qquad\qquad \land u \in s \underset{\mathsf{cs}^\checkmark}{\parallel} t \end{array} \right\}$$

(9.1)

$$\cup \left\{ \begin{array}{l} (u, Y \cup Z) \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \qquad \land \exists\, s, t \bullet (s, Y) \in \{((tr' - tr) \frown \langle \checkmark \rangle, ref') \mid okay \land \neg\, wait' \land (P)^t_f\} \\ \qquad\qquad \land (t, Z) \in \{(tr' - tr, ref') \mid okay \land wait' \land (Q)^t_f\} \\ \qquad\qquad \land u \in s \underset{\mathsf{cs}^\checkmark}{\parallel} t \end{array} \right\}$$

(9.2)

$$\cup \left\{ \begin{array}{l} (u, Y \cup Z) \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \quad \wedge \exists\, s, t \bullet (s, Y) \in \{((tr' - tr) \frown \langle \checkmark \rangle, ref') \mid okay \wedge \neg\, wait' \wedge (P)^{t}_{f}\} \\ \qquad \wedge (t, Z) \in \{(tr' - tr, ref') \mid okay \wedge \neg\, wait' \wedge (Q)^{t}_{f}\} \\ \qquad \wedge u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array} \right\}$$

(10)

$$\cup \left\{ \begin{array}{l} (u, Y \cup Z \cup \{\checkmark\}) \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \quad \wedge \exists\, s, t \bullet (s, Y) \in \{((tr' - tr) \frown \langle \checkmark \rangle, ref') \mid okay \wedge \neg\, wait' \wedge (P)^{t}_{f} \\ \qquad \wedge (t, Z) \in \{(tr' - tr, ref') \mid okay \wedge wait' \wedge (Q)^{t}_{f}\} \\ \qquad \wedge u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array} \right.$$

(11)

$$\cup \left\{ \begin{array}{l} (u, Y \cup Z) \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \quad \wedge \exists\, s, t \bullet (s, Y) \in \{((tr' - tr) \frown \langle \checkmark \rangle, ref') \mid okay \wedge \neg\, wait' \wedge (P)^{t}_{f}\} \\ \qquad \wedge (t, Z) \in \{((tr' - tr) \frown \langle \checkmark \rangle, ref') \mid okay \wedge \neg\, wait' \wedge (Q)^{t}_{f}\} \\ \qquad \wedge u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array} \right\}$$

(12)

$$\cup \left\{ \begin{array}{l} (u, Y \cup Z \cup \{\checkmark\}) \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \quad \wedge \exists\, s, t \bullet (s, Y) \in \{((tr' - tr) \frown \langle \checkmark \rangle, ref') \mid okay \wedge \neg\, wait' \wedge (P)^{t}_{f} \\ \qquad \wedge (t, Z) \in \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref') \\ \mid okay \wedge \neg\, wait' \wedge (Q)^{t}_{f} \end{array} \right\} \\ \qquad \wedge u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array} \right.$$

(13.1)

$$\cup \left\{ \begin{array}{l} (u, Y \cup Z \cup \{\checkmark\}) \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \quad \wedge \exists\, s, t \bullet (s, Y) \in \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref') \\ \mid okay \wedge \neg\, wait' \wedge (P)^{t}_{f} \end{array} \right\} \\ \qquad \wedge (t, Z) \in \{(tr' - tr, ref') \mid okay \wedge wait' \wedge (Q)^{t}_{f}\} \\ \qquad \wedge u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array} \right\}$$

(13.2)

$$\cup \left\{ \begin{array}{l} (u, Y \cup Z \cup \{\checkmark\}) \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \quad \wedge \exists\, s, t \bullet (s, Y) \in \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref') \\ \mid okay \wedge \neg\, wait' \wedge (P)^{t}_{f} \end{array} \right\} \\ \qquad \wedge (t, Z) \in \{(tr' - tr, ref') \mid okay \wedge \neg\, wait' \wedge (Q)^{t}_{f}\} \\ \qquad \wedge u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array} \right\}$$

(14)

$$\cup \left\{ \begin{array}{l} (u, Y \cup Z \cup \{\checkmark\}) \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \quad \land \exists\, s, t \bullet (s, Y) \in \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref') \\ \mid okay \land \neg\, wait' \land (P)^t_f \end{array} \right\} \\ \quad\quad \land (t, Z) \in \{(tr' - tr, ref') \mid okay \land wait' \land (Q)^t_f\} \\ \quad\quad \land u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array} \right\}$$

(15)

$$\cup \left\{ \begin{array}{l} (u, Y \cup Z \cup \{\checkmark\}) \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \quad \land \exists\, s, t \bullet (s, Y) \in \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref') \\ \mid okay \land \neg\, wait' \land (P)^t_f \end{array} \right\} \\ \quad\quad \land (t, Z) \in \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref') \\ \mid okay \land \neg\, wait' \land (Q)^t_f \end{array} \right\} \\ \quad\quad \land u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array} \right\}$$

(16)

$$\cup \left\{ \begin{array}{l} (u, Y \cup Z \cup \{\checkmark\}) \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \quad \land \exists\, s, t \bullet (s, Y) \in \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref') \\ \mid okay \land \neg\, wait' \land (P)^t_f \end{array} \right\} \\ \quad\quad \land (t, Z) \in \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref') \\ \mid okay \land \neg\, wait' \land (Q)^t_f \end{array} \right\} \\ \quad\quad \land u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array} \right\}$$

[ST and SC (refusals on (2.*), (4.*), (5.*), (6), (7), (8), (10), (12) and (13) to (16)]

$=$

(1.1)

$$\left\{ \begin{array}{l} (u, Y \cup Z) \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \quad \land \exists\, s, t \bullet (s, Y) \in \{(tr' - tr, ref') \mid okay \land wait' \land (P)^t_f\} \\ \quad\quad \land (t, Z) \in \{(tr' - tr, ref') \mid okay \land wait' \land (Q)^t_f\} \\ \quad\quad \land u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array} \right\}$$

(1.2)

$$\cup \left\{ \begin{array}{l} (u, Y \cup Z) \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \quad \land \exists\, s, t \bullet (s, Y) \in \{(tr' - tr, ref') \mid okay \land wait' \land (P)^t_f\} \\ \quad\quad \land (t, Z) \in \{(tr' - tr, ref') \mid okay \land \neg\, wait' \land (Q)^t_f\} \\ \quad\quad \land u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array} \right\}$$

(1.3)

$$\cup \left\{ \begin{array}{l} (u, Y \cup Z) \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \quad \wedge \exists s, t \bullet (s, Y) \in \{(tr' - tr, ref') \mid okay \wedge \neg wait' \wedge (P)^t_f\} \\ \quad\quad \wedge (t, Z) \in \{(tr' - tr, ref') \mid okay \wedge wait' \wedge (Q)^t_f\} \\ \quad\quad \wedge u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array} \right\}$$

(1.4)

$$\cup \left\{ \begin{array}{l} (u, Y \cup Z) \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \quad \wedge \exists s, t \bullet (s, Y) \in \{(tr' - tr, ref') \mid okay \wedge \neg wait' \wedge (P)^t_f\} \\ \quad\quad \wedge (t, Z) \in \{(tr' - tr, ref') \mid okay \wedge \neg wait' \wedge (Q)^t_f\} \\ \quad\quad \wedge u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array} \right\}$$

(2.1)

$$\cup \left\{ \begin{array}{l} (u, Y \cup Z) \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \quad \wedge \exists s, t \bullet (s, Y) \in \{(tr' - tr, ref') \mid okay \wedge wait' \wedge (P)^t_f\} \\ \quad\quad \wedge (t, Z) \in \{(tr' - tr, ref' \cup \{\checkmark\}) \mid okay \wedge wait' \wedge (Q)^t_f\} \\ \quad\quad \wedge u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array} \right\}$$

(2.2)

$$\cup \left\{ \begin{array}{l} (u, Y \cup Z) \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \quad \wedge \exists s, t \bullet (s, Y) \in \{(tr' - tr, ref') \mid okay \wedge \neg wait' \wedge (P)^t_f\} \\ \quad\quad \wedge (t, Z) \in \{(tr' - tr, ref' \cup \{\checkmark\}) \mid okay \wedge wait' \wedge (Q)^t_f\} \\ \quad\quad \wedge u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array} \right\}$$

(3.1)

$$\cup \left\{ \begin{array}{l} (u, Y \cup Z) \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \quad \wedge \exists s, t \bullet (s, Y) \in \{(tr' - tr, ref') \mid okay \wedge wait' \wedge (P)^t_f\} \\ \quad\quad \wedge (t, Z) \in \{((tr' - tr) \frown \langle\checkmark\rangle, ref') \mid okay \wedge \neg wait' \wedge (Q)^t_f\} \\ \quad\quad \wedge u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array} \right\}$$

(3.2)

$$\cup \left\{ \begin{array}{l} (u, Y \cup Z) \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \quad \wedge \exists s, t \bullet (s, Y) \in \{(tr' - tr, ref') \mid okay \wedge \neg wait' \wedge (P)^t_f\} \\ \quad\quad \wedge (t, Z) \in \{((tr' - tr) \frown \langle\checkmark\rangle, ref') \mid okay \wedge \neg wait' \wedge (Q)^t_f\} \\ \quad\quad \wedge u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array} \right\}$$

(4.1)

$$\cup \left\{ \begin{array}{l} (u, Y \cup Z) \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \quad \wedge \exists\, s, t \bullet (s, Y) \in \{(tr' - tr, ref') \mid okay \wedge wait' \wedge (P)^t_f\} \\ \quad\quad \wedge (t, Z) \in \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref' \cup \{\checkmark\}) \\ \mid okay \wedge \neg\, wait' \wedge (Q)^t_f \end{array} \right\} \\ \quad\quad \wedge u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array} \right\}$$

(4.2)

$$\cup \left\{ \begin{array}{l} (u, Y \cup Z) \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \quad \wedge \exists\, s, t \bullet (s, Y) \in \{(tr' - tr, ref') \mid okay \wedge \neg\, wait' \wedge (P)^t_f\} \\ \quad\quad \wedge (t, Z) \in \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref' \cup \{\checkmark\}) \\ \mid okay \wedge \neg\, wait' \wedge (Q)^t_f \end{array} \right\} \\ \quad\quad \wedge u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array} \right\}$$

(5.1)

$$\cup \left\{ \begin{array}{l} (u, Y \cup Z) \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \quad \wedge \exists\, s, t \bullet (s, Y) \in \{(tr' - tr, ref' \cup \{\checkmark\}) \mid okay \wedge wait' \wedge (P)^t_f\} \\ \quad\quad \wedge (t, Z) \in \{(tr' - tr, ref') \mid okay \wedge wait' \wedge (Q)^t_f\} \\ \quad\quad \wedge u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array} \right\}$$

(5.2)

$$\cup \left\{ \begin{array}{l} (u, Y \cup Z) \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \quad \wedge \exists\, s, t \bullet (s, Y) \in \{(tr' - tr, ref' \cup \{\checkmark\}) \mid okay \wedge wait' \wedge (P)^t_f\} \\ \quad\quad \wedge (t, Z) \in \{(tr' - tr, ref') \mid okay \wedge \neg\, wait' \wedge (Q)^t_f\} \\ \quad\quad \wedge u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array} \right\}$$

(6)

$$\cup \left\{ \begin{array}{l} (u, Y \cup Z) \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \quad \wedge \exists\, s, t \bullet (s, Y) \in \{(tr' - tr, ref' \cup \{\checkmark\}) \mid okay \wedge wait' \wedge (P)^t_f\} \\ \quad\quad \wedge (t, Z) \in \{(tr' - tr, ref' \cup \{\checkmark\}) \mid okay \wedge wait' \wedge (Q)^t_f\} \\ \quad\quad \wedge u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array} \right\}$$

(7)

$$\cup \left\{ \begin{array}{l} (u, Y \cup Z) \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \quad \wedge \exists\, s, t \bullet (s, Y) \in \{(tr' - tr, ref' \cup \{\checkmark\}) \mid okay \wedge wait' \wedge (P)^t_f\} \\ \quad\quad \wedge (t, Z) \in \{((tr' - tr) \frown \langle \checkmark \rangle, ref') \mid okay \wedge \neg\, wait' \wedge (Q)^t_f\} \\ \quad\quad \wedge u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array} \right\}$$

(8)

$$\cup \left\{ \begin{array}{l} (u, Y \cup Z) \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \quad \wedge \exists\, s, t \bullet (s, Y) \in \{(tr' - tr, ref' \cup \{\checkmark\}) \mid okay \wedge wait' \wedge (P)^t_f\} \\ \qquad \wedge (t, Z) \in \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref' \cup \{\checkmark\}) \\ \mid okay \wedge \neg\, wait' \wedge (Q)^t_f \end{array} \right\} \\ \qquad \wedge u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array} \right\}$$

(9.1)

$$\cup \left\{ \begin{array}{l} (u, Y \cup Z) \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \quad \wedge \exists\, s, t \bullet (s, Y) \in \{((tr' - tr) \frown \langle \checkmark \rangle, ref') \mid okay \wedge \neg\, wait' \wedge (P)^t_f\} \\ \qquad \wedge (t, Z) \in \{(tr' - tr, ref') \mid okay \wedge wait' \wedge (Q)^t_f\} \\ \qquad \wedge u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array} \right\}$$

(9.2)

$$\cup \left\{ \begin{array}{l} (u, Y \cup Z) \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \quad \wedge \exists\, s, t \bullet (s, Y) \in \{((tr' - tr) \frown \langle \checkmark \rangle, ref') \mid okay \wedge \neg\, wait' \wedge (P)^t_f\} \\ \qquad \wedge (t, Z) \in \{(tr' - tr, ref') \mid okay \wedge \neg\, wait' \wedge (Q)^t_f\} \\ \qquad \wedge u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array} \right\}$$

(10)

$$\cup \left\{ \begin{array}{l} (u, Y \cup Z) \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \quad \wedge \exists\, s, t \bullet (s, Y) \in \{((tr' - tr) \frown \langle \checkmark \rangle, ref') \mid okay \wedge \neg\, wait' \wedge (P)^t_f\} \\ \qquad \wedge (t, Z) \in \{(tr' - tr, ref' \cup \{\checkmark\}) \mid okay \wedge wait' \wedge (Q)^t_f\} \\ \qquad \wedge u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array} \right\}$$

(11)

$$\cup \left\{ \begin{array}{l} (u, Y \cup Z) \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \quad \wedge \exists\, s, t \bullet (s, Y) \in \{((tr' - tr) \frown \langle \checkmark \rangle, ref') \mid okay \wedge \neg\, wait' \wedge (P)^t_f\} \\ \qquad \wedge (t, Z) \in \{((tr' - tr) \frown \langle \checkmark \rangle, ref') \mid okay \wedge \neg\, wait' \wedge (Q)^t_f\} \\ \qquad \wedge u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array} \right\}$$

(12)

$$\cup \left\{ \begin{array}{l} (u, Y \cup Z) \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \quad \wedge \exists\, s, t \bullet (s, Y) \in \{((tr' - tr) \frown \langle \checkmark \rangle, ref') \mid okay \wedge \neg\, wait' \wedge (P)^t_f\} \\ \qquad \wedge (t, Z) \in \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref' \cup \{\checkmark\}) \\ \mid okay \wedge \neg\, wait' \wedge (Q)^t_f \end{array} \right\} \\ \qquad \wedge u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array} \right\}$$

(13.1)

$$\cup \left\{ \begin{array}{l} (u, Y \cup Z) \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \qquad \wedge \exists\, s, t \bullet (s, Y) \in \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref' \cup \{\checkmark\}) \\ \mid okay \wedge \neg\, wait' \wedge (P)^t_f \end{array} \right\} \\ \qquad\qquad \wedge (t, Z) \in \{(tr' - tr, ref') \mid okay \wedge wait' \wedge (Q)^t_f\} \\ \qquad\qquad \wedge u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array} \right\}$$

(13.2)

$$\cup \left\{ \begin{array}{l} (u, Y \cup Z) \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \qquad \wedge \exists\, s, t \bullet (s, Y) \in \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref' \cup \{\checkmark\}) \\ \mid okay \wedge \neg\, wait' \wedge (P)^t_f \end{array} \right\} \\ \qquad\qquad \wedge (t, Z) \in \{(tr' - tr, ref') \mid okay \wedge \neg\, wait' \wedge (Q)^t_f\} \\ \qquad\qquad \wedge u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array} \right\}$$

(14)

$$\cup \left\{ \begin{array}{l} (u, Y \cup Z) \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \qquad \wedge \exists\, s, t \bullet (s, Y) \in \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref' \cup \{\checkmark\}) \\ \mid okay \wedge \neg\, wait' \wedge (P)^t_f \end{array} \right\} \\ \qquad\qquad \wedge (t, Z) \in \{(tr' - tr, ref' \cup \{\checkmark\}) \mid okay \wedge wait' \wedge (Q)^t_f\} \\ \qquad\qquad \wedge u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array} \right\}$$

(15)

$$\cup \left\{ \begin{array}{l} (u, Y \cup Z) \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \qquad \wedge \exists\, s, t \bullet (s, Y) \in \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref' \cup \{\checkmark\}) \\ \mid okay \wedge \neg\, wait' \wedge (P)^t_f \end{array} \right\} \\ \qquad\qquad \wedge (t, Z) \in \{((tr' - tr) \frown \langle \checkmark \rangle, ref') \mid okay \wedge \neg\, wait' \wedge (Q)^t_f\} \\ \qquad\qquad \wedge u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array} \right\}$$

(16)

$$\cup \left\{ \begin{array}{l} (u, Y \cup Z) \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \qquad \wedge \exists\, s, t \bullet (s, Y) \in \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref' \cup \{\checkmark\}) \\ \mid okay \wedge \neg\, wait' \wedge (P)^t_f \end{array} \right\} \\ \qquad\qquad \wedge (t, Z) \in \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref' \cup \{\checkmark\}) \\ \mid okay \wedge \neg\, wait' \wedge (Q)^t_f \end{array} \right\} \\ \qquad\qquad \wedge u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array} \right\}$$

[ST, SC and PC on (1), (2), (3), (4), (5), (9), (13)]

$=$

(1)

$$\left\{\begin{array}{l} (u, Y \cup Z) \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \qquad \wedge \exists\, s, t \bullet (s, Y) \in \{(tr' - tr, ref') \mid okay \wedge (P)^t_f\} \\ \qquad\qquad \wedge (t, Z) \in \{(tr' - tr, ref') \mid okay \wedge (Q)^t_f\} \\ \qquad\qquad \wedge u \in s \underset{\mathsf{cs}^{\checkmark}}{\|} t \end{array}\right\}$$

(2)

$$\cup \left\{\begin{array}{l} (u, Y \cup Z) \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \qquad \wedge \exists\, s, t \bullet (s, Y) \in \{(tr' - tr, ref') \mid okay \wedge (P)^t_f\} \\ \qquad\qquad \wedge (t, Z) \in \{(tr' - tr, ref' \cup \{\checkmark\}) \mid okay \wedge wait' \wedge (Q)^t_f\} \\ \qquad\qquad \wedge u \in s \underset{\mathsf{cs}^{\checkmark}}{\|} t \end{array}\right\}$$

(3)

$$\cup \left\{\begin{array}{l} (u, Y \cup Z) \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \qquad \wedge \exists\, s, t \bullet (s, Y) \in \{(tr' - tr, ref') \mid okay \wedge (P)^t_f\} \\ \qquad\qquad \wedge (t, Z) \in \{((tr' - tr) \frown \langle\checkmark\rangle, ref') \mid okay \wedge \neg\, wait' \wedge (Q)^t_f\} \\ \qquad\qquad \wedge u \in s \underset{\mathsf{cs}^{\checkmark}}{\|} t \end{array}\right\}$$

(4)

$$\cup \left\{\begin{array}{l} (u, Y \cup Z) \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \qquad \wedge \exists\, s, t \bullet (s, Y) \in \{(tr' - tr, ref') \mid okay \wedge (P)^t_f\} \\ \qquad\qquad \wedge (t, Z) \in \left\{\begin{array}{l} ((tr' - tr) \frown \langle\checkmark\rangle, ref' \cup \{\checkmark\}) \\ \mid okay \wedge \neg\, wait' \wedge (Q)^t_f \end{array}\right\} \\ \qquad\qquad \wedge u \in s \underset{\mathsf{cs}^{\checkmark}}{\|} t \end{array}\right\}$$

(5)

$$\cup \left\{\begin{array}{l} (u, Y \cup Z) \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \qquad \wedge \exists\, s, t \bullet (s, Y) \in \{(tr' - tr, ref' \cup \{\checkmark\}) \mid okay \wedge wait' \wedge (P)^t_f\} \\ \qquad\qquad \wedge (t, Z) \in \{(tr' - tr, ref') \mid okay \wedge (Q)^t_f\} \\ \qquad\qquad \wedge u \in s \underset{\mathsf{cs}^{\checkmark}}{\|} t \end{array}\right\}$$

(6)

$$\cup \left\{\begin{array}{l} (u, Y \cup Z) \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \qquad \wedge \exists\, s, t \bullet (s, Y) \in \{(tr' - tr, ref' \cup \{\checkmark\}) \mid okay \wedge wait' \wedge (P)^t_f\} \\ \qquad\qquad \wedge (t, Z) \in \{(tr' - tr, ref' \cup \{\checkmark\}) \mid okay \wedge wait' \wedge (Q)^t_f\} \\ \qquad\qquad \wedge u \in s \underset{\mathsf{cs}^{\checkmark}}{\|} t \end{array}\right\}$$

(7)

$$\cup \left\{ \begin{array}{l} (u, Y \cup Z) \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \quad \wedge \exists\, s, t \bullet (s, Y) \in \{(tr' - tr, ref' \cup \{\checkmark\}) \mid okay \wedge wait' \wedge (P)^t_f\} \\ \qquad \wedge (t, Z) \in \{((tr' - tr) ^\frown \langle \checkmark \rangle, ref') \mid okay \wedge \neg\, wait' \wedge (Q)^t_f\} \\ \qquad \wedge u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array} \right\}$$

(8)

$$\cup \left\{ \begin{array}{l} (u, Y \cup Z) \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \quad \wedge \exists\, s, t \bullet (s, Y) \in \{(tr' - tr, ref' \cup \{\checkmark\}) \mid okay \wedge wait' \wedge (P)^t_f\} \\ \qquad \wedge (t, Z) \in \left\{ \begin{array}{l} ((tr' - tr) ^\frown \langle \checkmark \rangle, ref' \cup \{\checkmark\}) \\ \mid okay \wedge \neg\, wait' \wedge (Q)^t_f \end{array} \right\} \\ \qquad \wedge u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array} \right\}$$

(9)

$$\cup \left\{ \begin{array}{l} (u, Y \cup Z) \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \quad \wedge \exists\, s, t \bullet (s, Y) \in \{((tr' - tr) ^\frown \langle \checkmark \rangle, ref') \mid okay \wedge \neg\, wait' \wedge (P)^t_f\} \\ \qquad \wedge (t, Z) \in \{(tr' - tr, ref') \mid okay \wedge (Q)^t_f\} \\ \qquad \wedge u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array} \right\}$$

(10)

$$\cup \left\{ \begin{array}{l} (u, Y \cup Z) \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \quad \wedge \exists\, s, t \bullet (s, Y) \in \{((tr' - tr) ^\frown \langle \checkmark \rangle, ref') \mid okay \wedge \neg\, wait' \wedge (P)^t_f\} \\ \qquad \wedge (t, Z) \in \{(tr' - tr, ref' \cup \{\checkmark\}) \mid okay \wedge wait' \wedge (Q)^t_f\} \\ \qquad \wedge u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array} \right\}$$

(11)

$$\cup \left\{ \begin{array}{l} (u, Y \cup Z) \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \quad \wedge \exists\, s, t \bullet (s, Y) \in \{((tr' - tr) ^\frown \langle \checkmark \rangle, ref') \mid okay \wedge \neg\, wait' \wedge (P)^t_f\} \\ \qquad \wedge (t, Z) \in \{((tr' - tr) ^\frown \langle \checkmark \rangle, ref') \mid okay \wedge \neg\, wait' \wedge (Q)^t_f\} \\ \qquad \wedge u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array} \right\}$$

(12)

$$\cup \left\{ \begin{array}{l} (u, Y \cup Z) \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \quad \wedge \exists\, s, t \bullet (s, Y) \in \{((tr' - tr) ^\frown \langle \checkmark \rangle, ref') \mid okay \wedge \neg\, wait' \wedge (P)^t_f\} \\ \qquad \wedge (t, Z) \in \left\{ \begin{array}{l} ((tr' - tr) ^\frown \langle \checkmark \rangle, ref' \cup \{\checkmark\}) \\ \mid okay \wedge \neg\, wait' \wedge (Q)^t_f \end{array} \right\} \\ \qquad \wedge u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array} \right\}$$

(13)

$$\cup \left\{ \begin{array}{l} (u, Y \cup Z) \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \qquad \land \exists\, s, t \bullet (s, Y) \in \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref' \cup \{\checkmark\}) \\ \mid okay \land \neg\, wait' \land (P)^t_f \end{array} \right\} \\ \qquad\qquad \land (t, Z) \in \{(tr' - tr, ref') \mid okay \land (Q)^t_f\} \\ \qquad\qquad \land u \in s \parallel_{\mathsf{cs}^{\checkmark}} t \end{array} \right\}$$

(14)

$$\cup \left\{ \begin{array}{l} (u, Y \cup Z) \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \qquad \land \exists\, s, t \bullet (s, Y) \in \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref' \cup \{\checkmark\}) \\ \mid okay \land \neg\, wait' \land (P)^t_f \end{array} \right\} \\ \qquad\qquad \land (t, Z) \in \{(tr' - tr, ref' \cup \{\checkmark\}) \mid okay \land wait' \land (Q)^t_f\} \\ \qquad\qquad \land u \in s \parallel_{\mathsf{cs}^{\checkmark}} t \end{array} \right\}$$

(15)

$$\cup \left\{ \begin{array}{l} (u, Y \cup Z) \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \qquad \land \exists\, s, t \bullet (s, Y) \in \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref' \cup \{\checkmark\}) \\ \mid okay \land \neg\, wait' \land (P)^t_f \end{array} \right\} \\ \qquad\qquad \land (t, Z) \in \{((tr' - tr) \frown \langle \checkmark \rangle, ref') \mid okay \land \neg\, wait' \land (Q)^t_f\} \\ \qquad\qquad \land u \in s \parallel_{\mathsf{cs}^{\checkmark}} t \end{array} \right\}$$

(16)

$$\cup \left\{ \begin{array}{l} (u, Y \cup Z) \mid Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \qquad \land \exists\, s, t \bullet (s, Y) \in \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref' \cup \{\checkmark\}) \\ \mid okay \land \neg\, wait' \land (P)^t_f \end{array} \right\} \\ \qquad\qquad \land (t, Z) \in \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref' \cup \{\checkmark\}) \\ \mid okay \land \neg\, wait' \land (Q)^t_f \end{array} \right\} \\ \qquad\qquad \land u \in s \parallel_{\mathsf{cs}^{\checkmark}} t \end{array} \right\}$$

[ST, SC and PC]

$$
= \left\{ \begin{array}{l}
(u, Y \cup Z) \mid \\
\quad Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\
\quad \land \exists\, s, t \bullet (s, Y) \in \left\{ \begin{array}{l} (tr' - tr, ref') \\ \mid okay \land (P)^t_f \end{array} \right\} \\
\qquad \cup \left\{ \begin{array}{l} (tr' - tr, ref' \cup \{\checkmark\}) \\ \mid okay \land wait' \land (P)^t_f \end{array} \right\} \\
\qquad \cup \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref') \\ \mid okay \land \neg\, wait' \land (P)^t_f \end{array} \right\} \\
\qquad \cup \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref' \cup \{\checkmark\}) \\ \mid okay \land \neg\, wait' \land (P)^t_f \end{array} \right\} \\
\qquad \land (t, Z) \in \left\{ \begin{array}{l} (tr' - tr, ref') \\ \mid okay \land (Q)^t_f \end{array} \right\} \\
\qquad \cup \left\{ \begin{array}{l} (tr' - tr, ref' \cup \{\checkmark\}) \\ \mid okay \land wait' \land (Q)^t_f \end{array} \right\} \\
\qquad \cup \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref') \\ \mid okay \land \neg\, wait' \land (Q)^t_f \end{array} \right\} \\
\qquad \cup \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref' \cup \{\checkmark\}) \\ \mid okay \land \neg\, wait' \land (Q)^t_f \end{array} \right\} \\
\qquad \land u \in s \underset{\mathsf{cs}^{\checkmark}}{\|} t
\end{array} \right\}
$$

[PC]

$$
= \left\{ \begin{array}{l}
(u, Y \cup Z) \mid \\
\quad Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\
\quad \land \exists\, s, t \bullet (s, Y) \in \left\{ \begin{array}{l} (tr' - tr, ref') \\ \mid okay \land \neg\, wait \land okay' \land (P) \end{array} \right\} \\
\qquad \cup \left\{ \begin{array}{l} (tr' - tr, ref' \cup \{\checkmark\}) \\ \mid okay \land \neg\, wait \land okay' \land wait' \land (P) \end{array} \right\} \\
\qquad \cup \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref') \\ \mid okay \land \neg\, wait \land okay' \land \neg\, wait' \land (P) \end{array} \right\} \\
\qquad \cup \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref' \cup \{\checkmark\}) \\ \mid okay \land \neg\, wait \land okay' \land \neg\, wait' \land (P) \end{array} \right\} \\
\qquad \land (t, Z) \in \left\{ \begin{array}{l} (tr' - tr, ref') \\ \mid okay \land \neg\, wait \land okay' \land (Q) \end{array} \right\} \\
\qquad \cup \left\{ \begin{array}{l} (tr' - tr, ref' \cup \{\checkmark\}) \\ \mid okay \land \neg\, wait \land okay' \land wait' \land (Q) \end{array} \right\} \\
\qquad \cup \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref') \\ \mid okay \land \neg\, wait \land okay' \land \neg\, wait' \land (Q) \end{array} \right\} \\
\qquad \cup \left\{ \begin{array}{l} ((tr' - tr) \frown \langle \checkmark \rangle, ref' \cup \{\checkmark\}) \\ \mid okay \land \neg\, wait \land okay' \land \neg\, wait' \land (Q) \end{array} \right\} \\
\qquad \land u \in s \underset{\mathsf{cs}^{\checkmark}}{\|} t
\end{array} \right\}
$$

$$[A^n]$$

$$= \left\{ \begin{array}{l} (u, Y \cup Z) \mid \\ \quad Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \quad \land \exists\, s, t \bullet (s, Y) \in \{(tr' - tr, ref') \mid (P)^n\} \\ \qquad \cup \{(tr' - tr, ref' \cup \{\checkmark\}) \mid (P)^n \land wait'\} \\ \qquad \cup \{((tr' - tr) \frown \langle \checkmark \rangle, ref') \mid \neg\, wait' \land (P)^n\} \\ \qquad \cup \{((tr' - tr) \frown \langle \checkmark \rangle, ref' \cup \{\checkmark\}) \mid \neg\, wait' \land (P)^n\} \\ \qquad \land (t, Z) \in \{(tr' - tr, ref') \mid (Q)^n\} \\ \qquad \cup \{(tr' - tr, ref' \cup \{\checkmark\}) \mid (Q)^n \land wait'\} \\ \qquad \cup \{((tr' - tr) \frown \langle \checkmark \rangle, ref') \mid \neg\, wait' \land (Q)^n\} \\ \qquad \cup \{((tr' - tr) \frown \langle \checkmark \rangle, ref' \cup \{\checkmark\}) \mid \neg\, wait' \land (Q)^n\} \\ \qquad \land u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array} \right\}$$

$$[A^t]$$

$$= \left\{ \begin{array}{l} (u, Y \cup Z) \mid \\ \quad Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \quad \land \exists\, s, t \bullet (s, Y) \in \{(tr' - tr, ref') \mid (P)^n\} \\ \qquad \cup \{(tr' - tr, ref' \cup \{\checkmark\}) \mid (P)^n \land wait'\} \\ \qquad \cup \{((tr' - tr) \frown \langle \checkmark \rangle, ref') \mid (P)^t\} \\ \qquad \cup \{((tr' - tr) \frown \langle \checkmark \rangle, ref' \cup \{\checkmark\}) \mid (P)^t\} \\ \qquad \land (t, Z) \in \{(tr' - tr, ref') \mid (Q)^n\} \\ \qquad \cup \{(tr' - tr, ref' \cup \{\checkmark\}) \mid (Q)^n \land wait'\} \\ \qquad \cup \{((tr' - tr) \frown \langle \checkmark \rangle, ref') \mid (Q)^t\} \\ \qquad \cup \{((tr' - tr) \frown \langle \checkmark \rangle, ref' \cup \{\checkmark\}) \mid (Q)^t\} \\ \qquad \land u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array} \right\}$$

$$[failures^{\mathcal{UTP}}]$$

$$= \left\{ \begin{array}{l} (u, Y \cup Z) \mid \\ \quad Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \quad \land \exists\, s, t \bullet (s, Y) \in failures^{\mathcal{UTP}}(P) \\ \qquad \land (t, Z) \in failures^{\mathcal{UTP}}(Q) \\ \qquad \land u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array} \right\}$$

$$[\mathrm{IH}]$$

$$= \left\{ \begin{array}{l} (u, Y \cup Z) \mid \\ \quad Y \setminus (\mathsf{cs} \cup \{\checkmark\}) = Z \setminus (\mathsf{cs} \cup \{\checkmark\}) \\ \quad \land \exists\, s, t \bullet (s, Y) \in failures(\Upsilon(P)) \\ \qquad \land (t, Z) \in failures(\Upsilon(Q))) \\ \qquad \land u \in s \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \end{array} \right\}$$

$$
\begin{aligned}
& & [\mathit{failures}] \\
&= \mathit{failures}(\Upsilon(P) \underset{\mathtt{cs}^{\checkmark}}{\parallel} \Upsilon(Q)) & [\text{Notation } (\Upsilon_{\mathbb{P}^{cs}}(cs) = \mathtt{cs})] \\
&= \mathit{failures}(\Upsilon(P) \underset{\Upsilon_{\mathbb{P}}(cs)}{\parallel} \Upsilon(Q)) & [\Upsilon] \\
&= \mathit{failures}(\Upsilon(P \llbracket ns_1 \mid cs \mid ns_2 \rrbracket Q))
\end{aligned}
$$

Theorem J.37

$$\mathit{failures}^{\mathcal{UTP}}(\llbracket cs \rrbracket\, x : S \bullet \llbracket ns \rrbracket\, A) = \mathit{failures}(\Upsilon(\llbracket cs \rrbracket\, x : S \bullet \llbracket ns \rrbracket\, A))$$

provided

1. $\forall\, i : S \bullet A[v_i/x]$ *is* $\boldsymbol{R}$
2. $\forall\, i : S \bullet A[v_i/x]$ *is divergence-free*
3. $S \neq \{\}$

Proof. By induction on S

Inductive Hypothesis (A):

$$\forall\, i : S \bullet \mathit{failures}^{\mathcal{UTP}}(A[v_i/x]) = \mathit{failures}(\Upsilon(A)[v_i/x])$$

Base Case. $S = \{v\}$

Proof.

$$
\begin{aligned}
& \mathit{failures}^{\mathcal{UTP}}(\llbracket cs \rrbracket\, x : S \bullet \llbracket ns \rrbracket\, A) & [\text{Assumption}] \\
&= \mathit{failures}^{\mathcal{UTP}}(\llbracket cs \rrbracket\, x : \{v\} \bullet \llbracket ns \rrbracket\, A) & [\text{Indexed parallel}] \\
&= \mathit{failures}^{\mathcal{UTP}}(A[v/x]) & [\text{IH}] \\
&= \mathit{failures}(\Upsilon(A[v/x])) & [\text{Indexed parallel}] \\
&= \mathit{failures}(\Upsilon(\llbracket cs \rrbracket\, x : \{v\} \bullet \llbracket ns \rrbracket\, A)) & [\text{Assumption}] \\
&= \mathit{failures}(\Upsilon(\llbracket cs \rrbracket\, x : S \bullet \llbracket ns \rrbracket\, A))
\end{aligned}
$$

Inductive Hypothesis (S):

$$\mathit{failures}^{\mathcal{UTP}}(\llbracket cs \rrbracket\, x : S \bullet \llbracket ns \rrbracket\, A) = \mathit{failures}(\Upsilon(\llbracket cs \rrbracket\, x : S \bullet \llbracket ns \rrbracket\, A))$$

Inductive Step

$$\mathit{failures}^{\mathcal{UTP}}(\lbrack\!\lbrack cs \rbrack\!\rbrack\, x : S \cup \{v_i\} \bullet \lbrack\!\lbrack ns \rbrack\!\rbrack\, A) = \mathit{failures}(\Upsilon(\lbrack\!\lbrack cs \rbrack\!\rbrack\, x : S \cup \{v_i\} \bullet \lbrack\!\lbrack ns \rbrack\!\rbrack\, A))$$

Proof.

$$\begin{aligned}
&\mathit{failures}^{\mathcal{UTP}}(\lbrack\!\lbrack cs \rbrack\!\rbrack\, x : S \cup \{v_i\} \bullet \lbrack\!\lbrack ns \rbrack\!\rbrack\, A) && \text{[Indexed parallel]}\\
&= \mathit{failures}^{\mathcal{UTP}}(A[v_i/x] \lbrack\!\lbrack\, ns[v_i/x] \mid cs \mid \textstyle\bigcup_{v:S\setminus\{v_i\}} ns[v/x] \,\rbrack\!\rbrack\, (\lbrack\!\lbrack cs \rbrack\!\rbrack\, x : S \setminus \{v_i\} \bullet \lbrack\!\lbrack ns \rbrack\!\rbrack\, A))\\
& && \text{[Theorem J.36 (Provisos, IH-}A\text{ and IH-}S\text{)]}\\
&= \mathit{failures}(\Upsilon(A[v_i/x] \lbrack\!\lbrack\, ns[v_i/x] \mid cs \mid \textstyle\bigcup_{v:S\setminus\{v_i\}} ns[v/x] \,\rbrack\!\rbrack\, (\lbrack\!\lbrack cs \rbrack\!\rbrack\, x : S \setminus \{v_i\} \bullet \lbrack\!\lbrack ns \rbrack\!\rbrack\, A)))\\
& && \text{[Indexed parallel]}\\
&= \mathit{failures}(\Upsilon(\lbrack\!\lbrack cs \rbrack\!\rbrack\, x : S \cup \{v_i\} \bullet \lbrack\!\lbrack ns \rbrack\!\rbrack\, A))
\end{aligned}$$

Theorem J.38

$$\begin{aligned}
&\mathit{failures}^{\mathcal{UTP}}(P \parallel\![ns_1 \mid ns_2]\!\parallel Q)\\
&=\\
&\mathit{failures}(\Upsilon(P \parallel\![ns_1 \mid ns_2]\!\parallel Q))
\end{aligned}$$

provided

1. *P and Q are divergence-free*

Proof.

$$\begin{aligned}
&\mathit{failures}^{\mathcal{UTP}}(P \parallel\![ns_1 \mid ns_2]\!\parallel Q) && \text{[Law 29]}\\
&= \mathit{failures}^{\mathcal{UTP}}(P \lbrack\!\lbrack\, ns_1 \mid \emptyset \mid ns_2 \,\rbrack\!\rbrack\, Q) && \text{[Theorem J.36 (proviso)]}\\
&= \mathit{failures}(\Upsilon(P \lbrack\!\lbrack\, ns_1 \mid \emptyset \mid ns_2 \,\rbrack\!\rbrack\, Q)) && \text{[Law 29]}\\
&= \mathit{failures}(\Upsilon(P \parallel\![ns_1 \mid ns_2]\!\parallel Q))
\end{aligned}$$

Theorem J.39

$$\mathit{failures}^{\mathcal{UTP}}(|\!|\!|\, x : S \bullet \lbrack\!\lbrack ns \rbrack\!\rbrack\, A) = \mathit{failures}(\Upsilon(|\!|\!|\, x : S \bullet \lbrack\!\lbrack ns \rbrack\!\rbrack\, A))$$

provided

1. *$\forall\, i : S \bullet A[v_i/x]$ is* $\boldsymbol{R}$
2. *$\forall\, i : S \bullet A[v_i/x]$ is divergence-free*
3. $S \neq \{\}$

Proof.

$$\begin{array}{lr}
\mathit{failures}^{\mathcal{UTP}}(\mathbin{|||}\, x : S \bullet \llbracket ns \rrbracket\, A) & [\text{Law } 29] \\
= \mathit{failures}^{\mathcal{UTP}}(\llbracket \emptyset \rrbracket\, x : S \bullet \llbracket ns \rrbracket\, A)) & [\text{Theorem J.36 (proviso)}] \\
= \mathit{failures}(\Upsilon(\llbracket \emptyset \rrbracket\, x : S \bullet \llbracket ns \rrbracket\, A)) & [\text{Law } 29] \\
= \mathit{failures}(\Upsilon(\mathbin{|||}\, x : S \bullet \llbracket ns \rrbracket\, A))
\end{array}$$

Theorem J.40 $\mathit{traces}^{\mathcal{UTP}}(A \setminus cs) = \mathit{traces}(\Upsilon(A \setminus cs))$

Proof. Under development.

Theorem J.41 $\mathit{failures}^{\mathcal{UTP}}(P \setminus cs) = \mathit{failures}(\Upsilon(P \setminus cs))$

Proof. To be done.

Theorem J.42 $\mathit{traces}^{\mathcal{UTP}}(\mu X \bullet A(X)) = \mathit{traces}(\Upsilon(\mu X \bullet A(X)))$

Proof. To be done.

Theorem J.43 $\mathit{failures}^{\mathcal{UTP}}(\mu X \bullet P(X)) = \mathit{failures}(\Upsilon(\mu X \bullet P(X)))$

Proof. To be done.

Theorem J.44 $\mathit{traces}^{\mathcal{UTP}}(P[\mathit{old} := \mathit{new}]) = \mathit{traces}(\Upsilon(P[\mathit{old} := \mathit{new}]))$

Proof. To be done.

Theorem J.45 $\mathit{failures}^{\mathcal{UTP}}(P[\mathit{old} := \mathit{new}]) = \mathit{failures}(\Upsilon(P[\mathit{old} := \mathit{new}]))$

Proof. To be done.

J.5 Auxiliary Lemmas

Lemma J.46

$$\begin{array}{l}
s \in (X \parallel_{cs} Y) \\
\Leftrightarrow s - t \in (X - t \parallel_{cs} Y - t)
\end{array}$$

Lemma J.47

$$s \leq X \land s \leq Y \land t \in X \parallel_{cs} Y$$
$$\Rightarrow$$
$$s \leq t$$

Lemma J.48

$$(e \in cs \land e \notin \operatorname{ran}(s))$$
$$\Rightarrow$$
$$(s \underset{\mathsf{cs}}{\parallel} t \frown \langle e \rangle = t \frown \langle e \rangle \underset{\mathsf{cs}}{\parallel} s = \{\})$$

Proof. To be done.

Lemma J.49

$$\{x \frown \langle \checkmark \rangle \mid x \in s \underset{\mathsf{cs}}{\parallel} t\}$$
$$=$$
$$\{x \mid x \in s \frown \langle \checkmark \rangle \underset{\mathsf{cs}^{\checkmark}}{\parallel} t \frown \langle \checkmark \rangle\}$$

provided $\checkmark \notin \operatorname{ran}(s) \cup \operatorname{ran}(t)$

Proof. To be done.

Lemma J.50

$$s \parallel_{cs} t = s \underset{cs}{\parallel} t$$

Proof. To be done.

Lemma J.51

$$s \parallel_{cs} t = s \underset{cs \,\cup\, \{e\}}{\parallel} t$$

provided $e \notin \operatorname{ran}(s) \cup \operatorname{ran}(t)$

Proof. To be done.

Lemma J.52

$$s \underset{cs}{\parallel} t \neq \emptyset \Leftrightarrow s \upharpoonright cs = t \upharpoonright cs\}$$

Proof. To be done.

Lemma J.53 *Let* $Y = YZ \cup cs'$ *and* $Z = YZ \cup cs''$ *such that* $(cs' \cup cs'') \subseteq cs$ *and* $YZ \cap cs = \{\}$ *then* $(Y \cup Z) \cap cs = (cs' \cup cs'')$.

Proof.

$$
\begin{aligned}
& (Y \cup Z) \cap cs && \text{[by the hypothesis]}\\
&= (YZ \cup cs' \cup YZ \cup cs'') \cap cs \\
& && \text{[set theory: idempotence and comutativity of } \cup]\\
&= (YZ \cup (cs' \cup cs'')) \cap cs && \text{[set theory: distribution of } \cap \text{ over } \cup]\\
&= (YZ \cap cs) \cup ((cs' \cup cs'') \cap cs) && \text{[by the hypothesis]}\\
&= \{\} \cup (cs' \cup cs'') && \text{[set theory: identity of } \cup]\\
&= (cs' \cup cs'')
\end{aligned}
$$

Lemma J.54 *Let* $Y = YZ \cup cs'$ *and* $Z = YZ \cup cs''$ *such that* $(cs' \cup cs'') \subseteq cs$ *and* $YZ \cap cs = \{\}$ *then* $(Y \cap Z) \setminus cs = YZ$.

Proof.

$$
\begin{aligned}
& (Y \cap Z) \setminus cs && \text{[by the hypothesis]}\\
&= ((YZ \cup cs') \cap (YZ \cup cs'')) \cap cs && \text{[set theory: distribution of } \cap \text{ over } \cup]\\
&= ((YZ \cap YZ) \cup (YZ \cap cs'') \cup (cs' \cap YZ) \cup (cs' \cap cs'')) \cap cs \\
& && \text{[hypothesis and set theory (idempotence and identity of } \cup)]\\
&= (YZ \cup (cs' \cap cs'')) \cap cs \\
& && \text{[set theory (distribution of } \setminus \text{ over } \cup\text{, identity of } \cup) \text{ and hypothesis]}\\
&= YZ
\end{aligned}
$$

Lemma J.55

$$
\begin{aligned}
& \{Y, Z, cs, ref' \mid Y \setminus cs = Z \setminus cs \wedge ref' = Y \cup Z \bullet ref'\}\\
& =\\
& \{Y, Z, cs, ref' \mid Y \setminus cs = Z \setminus cs \wedge ref' = ((Y \cup Z) \cap cs) \cup ((Y \cap Z) \setminus cs) \bullet ref'\}
\end{aligned}
$$

Proof.

$$
\begin{aligned}
& \{Y, Z, cs, ref' \mid Y \setminus cs = Z \setminus cs \wedge ref' = Y \cup Z \bullet ref'\} && \text{[One point rule]}\\
&= \{Y, Z, cs, ref' \mid (\exists YZ \bullet YZ = Y \setminus cs \wedge YZ = Z \setminus cs) \wedge ref' = Y \cup Z \bullet ref'\}
\end{aligned}
$$

[One point rule and predicate calculus]

$$= \left\{ \begin{array}{l} Y, Z, cs, ref' \\ \mid \left(\begin{array}{l} \exists\, YZ, cs', cs'' \\ \mid cs' \cup cs'' \subseteq cs \land YZ \cap cs = \{\} \\ \bullet\ Y = YZ \cup cs' \land Z = YZ \cup cs'' \end{array} \right) \land ref' = Y \cup Z \\ \bullet\ ref' \end{array} \right\}$$

[substitution]

$$= \left\{ \begin{array}{l} Y, Z, cs, ref' \\ \mid \left(\begin{array}{l} \exists\, YZ, cs', cs'' \\ \mid cs' \cup cs'' \subseteq cs \land YZ \cap cs = \{\} \\ \bullet\ Y = YZ \cup cs' \land Z = YZ \cup cs'' \end{array} \right) \\ \quad \land ref' = YZ \cup cs' \cup cs'' \\ \bullet\ ref' \end{array} \right\}$$

[Lemma J.53]

$$= \left\{ \begin{array}{l} Y, Z, cs, ref' \\ \mid \left(\begin{array}{l} \exists\, YZ, cs', cs'' \\ \mid cs' \cup cs'' \subseteq cs \land YZ \cap cs = \{\} \\ \bullet\ Y = YZ \cup cs' \land Z = YZ \cup cs'' \end{array} \right) \\ \quad \land ref' = YZ \cup ((Y \cup Z) \cap cs) \\ \bullet\ ref' \end{array} \right\}$$

[Lemma J.54]

$$= \left\{ \begin{array}{l} Y, Z, cs, ref' \\ \mid \left(\begin{array}{l} \exists\, YZ, cs', cs'' \\ \mid cs' \cup cs'' \subseteq cs \land YZ \cap cs = \{\} \\ \bullet\ Y = YZ \cup cs' \land Z = YZ \cup cs'' \end{array} \right) \\ \quad \land ref' = ((X \cap Z) \setminus cs) \cup ((Y \cup Z) \cap cs) \\ \bullet\ ref' \end{array} \right\}$$

[one point rule and predicate calculus]

$$= \{ Y, Z, cs, ref' \mid (Y \setminus cs = Z \setminus cs) \land ref' = ((X \cap Z) \setminus cs) \cup ((Y \cup Z) \cap cs) \bullet ref' \}$$

K Proofs of the Rewrite from Stateful *Circus* into Stateless *Circus*

In this section, we demonstrate the correctness of the translation function Ω that rewrites stateful *Circus* processes into stateless *Circus* processes. The overall proof is by induction on the syntax of accepted actions. We consider

Skip, *Stop*, *Chaos*, prefixing, external and internal choice, guarded actions, sequence, hiding, alternation, and assignment.

K.1 Skip

Theorem K.1

$$\begin{array}{l} P_S.\mathit{Skip} \\ = \\ \Omega(P_S.\mathit{Skip}) \end{array}$$

Proof. In this proof and those that follow we will consider a single state component x. The generalisation of this proof by induction on the number of state components is rather simple, but omitted here for the sake of presentation.

$$\begin{array}{ll} \Omega(P_S.\mathit{Skip}) & [\Omega] \\ = & \\ P. & \\ \quad \mathbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \wedge \mathit{inv}(b(v_0))\} \bullet & \\ \quad \left(\begin{array}{l} (\Omega_A(\mathit{Skip});\ \mathit{terminate} \to \mathit{Skip}) \\ [\![\emptyset \mid \mathit{MEM}_I \mid \{b\}]\!] \\ \mathit{Memory}(b) \end{array}\right) \setminus \mathit{MEM}_I & \\ & [\Omega_A] \\ = P. & \\ \quad \mathbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \wedge \mathit{inv}(b(v_0))\} \bullet & \\ \quad \left(\begin{array}{l} (\mathit{Skip};\ \mathit{terminate} \to \mathit{Skip}) \\ [\![\emptyset \mid \mathit{MEM}_I \mid \{b\}]\!] \\ \mathit{Memory}(b) \end{array}\right) \setminus \mathit{MEM}_I & \\ & [\text{Law } 8] \\ = P. & \\ \quad \mathbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \wedge \mathit{inv}(b(v_0))\} \bullet & \\ \quad \left(\begin{array}{l} (\mathit{terminate} \to \mathit{Skip}) \\ [\![\emptyset \mid \mathit{MEM}_I \mid \{b\}]\!] \\ \mathit{Memory}(b) \end{array}\right) \setminus \mathit{MEM}_I & \\ & [\text{Lemma K.2}] \\ = P. & \end{array}$$

$$\begin{array}{l}
\mathbf{var}\ b : \{x : BINDING \mid b(v_0) \in T_0 \land inv(b(v_0))\} \bullet \\
\quad \left(\left(\begin{array}{l} (terminate \rightarrow Skip) \\ [\![\emptyset \mid MEM_I \mid \{b\}]\!] \\ \left(\begin{array}{l} (\Box\ get.n!b(n) \rightarrow Memory(b)) \\ \Box\ \left(\ \Box\ set.n?nv \rightarrow Memory(b \oplus \{n \mapsto nv\})\ \right) \\ \Box\ terminate \rightarrow Skip; \end{array} \right) \end{array} \right) \right) \\
\quad \setminus MEM_I
\end{array}$$

[Law 10]

provided

$\{terminate\} \subseteq MEM_I$

$\{get, set\} \subseteq MEM_I$

$\{get, set\} \cap \{terminate\} = \emptyset$

$= P.$

$$\begin{array}{l}
\mathbf{var}\ b : \{x : BINDING \mid b(v_0) \in T_0 \land inv(b(v_0))\} \bullet \\
\quad \left(\begin{array}{l} (terminate \rightarrow Skip) \\ [\![\emptyset \mid MEM_I \mid \{b\}]\!] \\ (terminate \rightarrow Skip) \end{array} \right) \setminus MEM_I
\end{array}$$

[Law 25]

provided

$[terminate \in MEM_I]$

$= P.$

$$\begin{array}{l}
\mathbf{var}\ b : \{x : BINDING \mid b(v_0) \in T_0 \land inv(b(v_0))\} \bullet \\
\quad \left(\begin{array}{l} Skip \\ [\![\emptyset \mid MEM_I \mid \{b\}]\!] \\ Skip \end{array} \right) \setminus MEM_I
\end{array}$$

[Law 28]

$= P.$

$$\begin{array}{l}
\mathbf{var}\ b : \{x : BINDING \mid b(v_0) \in T_0 \land inv(b(v_0))\} \bullet \\
\quad Skip \setminus MEM_I
\end{array}$$

[Law 15]

provided

$[MEM_I \cap usedC(Skip) = \emptyset]$

$= P.$

$$\begin{array}{l}
\mathbf{var}\ b : \{x : BINDING \mid b(v_0) \in T_0 \land inv(b(v_0))\} \bullet \\
\quad Skip
\end{array}$$

[Law 26(b is the only component of S)]

$= P_S.\mathit{Skip}$

□

K.2 Stop

Theorem K.2

$$P_S.\mathit{Stop} \\ = \\ \Omega(P_S.\mathit{Stop})$$

Proof.

$\Omega(P_S.\mathit{Stop})$ [Ω]

$$= P. \\ \quad \mathbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \wedge \mathit{inv}(b(v_0))\} \bullet \\ \quad \left(\begin{array}{l} (\Omega_A(\mathit{Stop});\ \mathit{terminate} \to \mathit{Skip}) \\ [\![\emptyset \mid \mathit{MEM}_I \mid \{b\}]\!] \\ \mathit{Memory}(b) \end{array}\right) \setminus \mathit{MEM}_I$$

[Ω_A]

$$= P. \\ \quad \mathbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \wedge \mathit{inv}(b(v_0))\} \bullet \\ \quad \left(\begin{array}{l} (\mathit{Stop};\ \mathit{terminate} \to \mathit{Skip}) \\ [\![\emptyset \mid \mathit{MEM}_I \mid \{b\}]\!] \\ \mathit{Memory}(b) \end{array}\right) \setminus \mathit{MEM}_I$$

[Law 22]

$$= P. \\ \quad \mathbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \wedge \mathit{inv}(b(v_0))\} \bullet \\ \quad \left(\begin{array}{l} \mathit{Stop} \\ [\![\emptyset \mid \mathit{MEM}_I \mid \{b\}]\!] \\ \mathit{Memory}(b) \end{array}\right) \setminus \mathit{MEM}_I$$

[Lemma K.2]

$= P.$

$$\begin{array}{l}\mathbf{var}\ b : \{x : BINDING \mid b(v_0) \in T_0 \land inv(b(v_0))\} \bullet \\ \quad \left(\left(\begin{array}{l} Stop \\ [\![\emptyset \mid MEM_I \mid \{b\}]\!] \\ \left(\begin{array}{l}(\Box\, n : \mathrm{dom}\, b \bullet get.n!b(n) \rightarrow Memory(b)) \\ \Box \left(\begin{array}{l}\Box\, n : \mathrm{dom}\, b \bullet \\ \quad set.n?nv \rightarrow Memory(b \oplus \{n \mapsto nv\})\end{array}\right) \\ \Box\ terminate \rightarrow Skip\end{array}\right)\end{array}\right)\right) \\ \quad \setminus MEM_I\end{array}$$

[Law 46]

provided

$\{get, set, terminate\} \subseteq MEM_I$

$= P.$

$$\begin{array}{l}\mathbf{var}\ b : \{x : BINDING \mid b(v_0) \in T_0 \land inv(b(v_0))\} \bullet \\ \quad Stop \setminus MEM_I\end{array}$$

[Law 15]

provided

$[MEM_I \cap usedC\{Stop\} = \emptyset]$

$= P.$

$$\begin{array}{l}\mathbf{var}\ b : \{x : BINDING \mid b(v_0) \in T_0 \land inv(b(v_0))\} \bullet \\ \quad Stop\end{array}$$

[Law 26(b is the only component of S)]

$= P_S.Stop$

□

K.3 Chaos

Theorem K.3

$$P_S.Chaos \\ = \\ \Omega(P_S.Chaos)$$

Proof.

$\Omega(P_S.Chaos)$ [Ω]

$= P.$

$$\mathbf{var}\ b : \{x : BINDING \mid b(v_0) \in T_0 \wedge inv(b(v_0))\} \bullet \left(\begin{array}{l}(\Omega_A(Chaos);\ terminate \to Skip)\\ \llbracket \emptyset \mid MEM_I \mid \{b\} \rrbracket\\ Memory(b)\end{array}\right) \setminus MEM_I$$

[Ω_A]

$$= P.\ \mathbf{var}\ b : \{x : BINDING \mid b(v_0) \in T_0 \wedge inv(b(v_0))\} \bullet \left(\begin{array}{l}(Chaos;\ terminate \to Skip)\\ \llbracket \emptyset \mid MEM_I \mid \{b\} \rrbracket\\ Memory(b)\end{array}\right) \setminus MEM_I$$

[Law 40]

$$= P.\ \mathbf{var}\ b : \{x : BINDING \mid b(v_0) \in T_0 \wedge inv(b(v_0))\} \bullet \left(\begin{array}{l}Chaos\\ \llbracket \emptyset \mid MEM_I \mid \{b\} \rrbracket\\ Memory(b)\end{array}\right) \setminus MEM_I$$

[Law 41]

$$= P.\ \mathbf{var}\ b : \{x : BINDING \mid b(v_0) \in T_0 \wedge inv(b(v_0))\} \bullet Chaos \setminus MEM_I$$

[Law 39]

$$= P.\ \mathbf{var}\ b : \{x : BINDING \mid b(v_0) \in T_0 \wedge inv(b(v_0))\} \bullet Chaos$$

[Law 26(b is the only component of S)]

$$= P_S.Chaos$$

□

K.4 Prefixing

Theorem K.4

$$P_S.(c \to A) = \Omega(P_S.(c \to A))$$

Proof. Inductive Hypothesis for any state S.

$$(\mathbf{vres}\ x : \mathit{BINDING} \bullet A(x))(b)$$
$$=$$
$$\left(\begin{array}{l}(\Omega_A(A);\ \mathit{terminate} \rightarrow \mathit{Skip})\\ [\![\emptyset \mid \mathit{MEM}_I \mid \{b\}]\!]\\ \mathit{Memory}(b)\end{array}\right) \setminus \mathit{MEM}_I$$

□

Proof.

$$\Omega(Ps.(c \rightarrow A))$$

[Ω]

$$= P.$$
$$\mathbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \wedge \mathit{inv}(b(v_0))\} \bullet$$
$$\left(\begin{array}{l}(\Omega_A(c \rightarrow A);\ \mathit{terminate} \rightarrow \mathit{Skip})\\ [\![\emptyset \mid \mathit{MEM}_I \mid \{b\}]\!]\\ \mathit{Memory}(b)\end{array}\right) \setminus \mathit{MEM}_I$$

[Ω_A]

$$= P.$$
$$\mathbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \wedge \mathit{inv}(b(v_0))\} \bullet$$
$$\left(\begin{array}{l}(c \rightarrow \Omega_A(A);\ \mathit{terminate} \rightarrow \mathit{Skip})\\ [\![\emptyset \mid \mathit{MEM}_I \mid \{b\}]\!]\\ \mathit{Memory}(b)\end{array}\right) \setminus \mathit{MEM}_I$$

[Law 34]

$$= P.$$
$$\mathbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \wedge \mathit{inv}(b(v_0))\} \bullet$$
$$\left(\begin{array}{l}((c \rightarrow \mathit{Skip});\Omega_A(A);\ \mathit{terminate} \rightarrow \mathit{Skip})\\ [\![\emptyset \mid \mathit{MEM}_I \mid \{b\}]\!]\\ \mathit{Memory}(b)\end{array}\right) \setminus \mathit{MEM}_I$$

[Law 30]

provided

$[\mathit{initials}(\mathit{Memory}(b)) \subseteq \mathit{MEM}_I]$

$[\mathit{MEM}_I \cap \mathit{usedC}(c \rightarrow \mathit{Skip}) = \emptyset]$

$[\mathit{wrtV}(c \rightarrow \mathit{Skip}) \cap \mathit{usedV}(\mathit{Memory}(b)) = \emptyset]$

$[\mathit{Memory}(b)$ is divergence-free$]$

$[\mathit{wrtV}(c \to \mathit{Skip}) \subseteq \emptyset]$

$= P.$

$$\mathbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \land \mathit{inv}(b(v_0))\} \bullet \left((c \to \mathit{Skip}); \begin{pmatrix} (\Omega_A(A);\ \mathit{terminate} \to \mathit{Skip}) \\ [\![\emptyset \mid \mathit{MEM}_I \mid \{b\}]\!] \\ \mathit{Memory}(b) \end{pmatrix} \right) \setminus \mathit{MEM}_I$$

[Law 31]

$= P.$

$$\mathbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \land \mathit{inv}(b(v_0))\} \bullet (c \to \mathit{Skip}) \setminus \mathit{MEM}_I; \begin{pmatrix} (\Omega_A(A);\ \mathit{terminate} \to \mathit{Skip}) \\ [\![\emptyset \mid \mathit{MEM}_I \mid \{b\}]\!] \\ \mathit{Memory}(b) \end{pmatrix} \setminus \mathit{MEM}_I$$

[Law 15]

provided

$[\mathit{MEM}_I \cap \mathit{usedC}(c \to \mathit{Skip}) = \emptyset)]$

$= P.$

$$\mathbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \land \mathit{inv}(b(v_0))\} \bullet (c \to \mathit{Skip}); \begin{pmatrix} \Omega_A(A);\ \mathit{terminate} \to \mathit{Skip} \\ [\![\emptyset \mid \mathit{MEM}_I \mid \{b\}]\!] \\ \mathit{Memory}(b) \end{pmatrix} \setminus \mathit{MEM}_I$$

[IH]

$= P.$

$$\mathbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \land \mathit{inv}(b(v_0))\} \bullet \begin{pmatrix} (c \to \mathit{Skip}); \\ (\mathbf{vres}\ x : \mathit{BINDING} \bullet A(x))(b) \end{pmatrix}$$

[Lemma K.1]

$= P.$

$$\mathbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \land \mathit{inv}(b(v_0))\} \bullet \begin{pmatrix} \mathbf{var}\ x : \mathit{BINDING} \bullet \\ \quad (c \to \mathit{Skip}); \\ \quad A(b) \end{pmatrix}$$

[Law 34]

$= P.$

$$\begin{array}{l}\textbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \land \mathit{inv}(b(v_0))\} \bullet \\ \quad \big(\ \textbf{var}\ x : \mathit{BINDING} \bullet c \to A(b)\ \big)\end{array}$$

[Law 6]

provided

$[x \notin FV(c \to A(b))]$

$= P.$

$$\begin{array}{l}\textbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \land \mathit{inv}(b(v_0))\} \bullet \\ \quad c \to A(b)\end{array}$$

[Law 26(b is the only component of S)]

$= P_S.(c \to A)$

□

K.5 Output Communications

Theorem K.5

$$\begin{array}{l} Ps.(c.e(b(v_0)) \to A) \\ = \\ \Omega(Ps.(c.e(b(v_0)) \to A)) \end{array}$$

Proof. Inductive Hypothesis for any state S.

$$\begin{array}{l} (\textbf{vres}\ x : \mathit{BINDING} \bullet A(x))(b) \\ = \\ \left(\begin{array}{l} (\Omega_A(A);\ \mathit{terminate} \to \mathit{Skip}) \\ [\![\emptyset \mid \mathit{MEM}_I \mid \{b\}]\!] \\ \mathit{Memory}(b) \end{array} \right) \setminus \mathit{MEM}_I \end{array}$$

Proof.

$\Omega(Ps.(c.e(b(v_0)) \to A))$

[Ω]

$= P.$

$$\begin{array}{l}\mathbf{var}\ b : \{x : BINDING \mid b(v_0) \in T_0 \land inv(b(v_0))\} \bullet \\ \left(\begin{array}{l}(\Omega_A(c.e(b(v_0)) \to A); \\ terminate \to Skip) \\ [\![\emptyset \mid MEM_I \mid \{b\}]\!] \\ Memory(b)\end{array}\right) \setminus MEM_I\end{array}$$

[Ω_A]

$= P.$

$$\begin{array}{l}\mathbf{var}\ b : \{x : BINDING \mid b(v_0) \in T_0 \land inv(b(v_0))\} \bullet \\ \left(\begin{array}{l}get.v_0?vv_0 \to c.e(vv_0) \to \Omega_A(A); \\ terminate \to Skip) \\ [\![\emptyset \mid MEM_I \mid \{b\}]\!] \\ Memory(b)\end{array}\right) \setminus MEM_I\end{array}$$

[Lemma K.2]

$= P.$

$$\begin{array}{l}\mathbf{var}\ b : \{x : BINDING \mid b(v_0) \in T_0 \land inv(b(v_0))\} \bullet \\ \left(\left(\begin{array}{l}\left(\begin{array}{l}get.v_0?vv_0 \to c.e(vv_0) \to \Omega_A(A); \\ terminate \to Skip\end{array}\right) \\ [\![\emptyset \mid MEM_I \mid \{b\}]\!] \\ \left(\begin{array}{l}(\Box\, get.n!b(n) \to Memory(b)) \\ \Box\ (\ \Box\, set.n?nv \to Memory(b \oplus \{n \mapsto nv\})\ ) \\ \Box\ terminate \to Skip;\end{array}\right)\end{array}\right)\right) \\ \setminus MEM_I\end{array}$$

[Law 10]

provided

$\{terminate\} \subseteq MEM_I$

$\{get, set\} \subseteq MEM_I$

$\{set, terminate\} \in MEM_I$

$\{set, terminate\} \notin \{get\}$

$= P.$

$$\begin{array}{l}\mathbf{var}\ b : \{x : BINDING \mid b(v_0) \in T_0 \land inv(b(v_0))\} \bullet \\ \left(\begin{array}{l}\left(\begin{array}{l}get.v_0?vv_0 \to c.e(vv_0) \to \Omega_A(A); \\ terminate \to Skip\end{array}\right) \\ [\![\emptyset \mid MEM_I \mid \{b\}]\!] \\ (\ get.n!b(n) \to Memory(b))\ )\end{array}\right) \setminus MEM_I\end{array}$$

[Law 24]

provided

$\{get\} \subseteq MEM_I$

$x \notin FV(get.n!b(n) \rightarrow Memory(b))$

$= P.$

$$\textbf{var } b : \{x : BINDING \mid b(v_0) \in T_0 \wedge inv(b(v_0))\} \bullet \left(\begin{array}{l} c.e(b(v_0)) \rightarrow \Omega_A(A); \\ terminate \rightarrow Skip \\ [\![\emptyset \mid MEM_I \mid \{b\}]\!] \\ Memory(b) \end{array}\right) \setminus MEM_I$$

[Law 30]

provided

$[initials(Memory(b)) \subseteq MEM_I]$

$[MEM_I \cap usedC(c.e(b(v_0)) \rightarrow Skip) = \emptyset]$

$[wrtV(c.e(b(v_0)) \rightarrow Skip) \cap usedV(Memory(b)) = \emptyset]$

$[Memory(b)$ is divergence-free$]$

$[wrtV(c.e(b(v_0)) \rightarrow Skip) \subseteq \emptyset]$

$=$

$P.$

$$\textbf{var } b : \{x : BINDING \mid b(v_0) \in T_0 \wedge inv(b(v_0))\} \bullet \left(\begin{array}{l} c.e(b(v_0)) \rightarrow Skip; \\ \left(\begin{array}{l} \Omega_A(A); terminate \rightarrow Skip \\ [\![\emptyset \mid MEM_I \mid \{b\}]\!] \\ Memory(b) \end{array}\right) \end{array}\right) \setminus MEM_I$$

[Law 31]

$= P.$

$$\textbf{var } b : \{x : BINDING \mid b(v_0) \in T_0 \wedge inv(b(v_0))\} \bullet \begin{array}{l} c.e(b(v_0)) \rightarrow Skip \setminus MEM_I; \\ \left(\begin{array}{l} (\Omega_A(A); terminate \rightarrow Skip) \\ [\![\emptyset \mid MEM_I \mid \{b\}]\!] \\ (Memory(b)) \end{array}\right) \setminus MEM_I \end{array}$$

[Law 15]

provided

$[MEM_I \cap usedC(c.e(b(v_0)) \rightarrow Skip) = \emptyset)]$

$=$

$P.$

$$\begin{array}{l}\textbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \land \mathit{inv}(b(v_0))\} \bullet \\ \quad c.e(b(v_0)) \rightarrow \mathit{Skip}; \\ \quad \left(\begin{array}{l}(\Omega_A(A);\ \mathit{terminate} \rightarrow \mathit{Skip}) \\ [\![\emptyset \mid \mathit{MEM}_I \mid \{b\}]\!] \\ (\mathit{Memory}(b))\end{array}\right) \setminus \mathit{MEM}_I\end{array}$$

[IH]

$$= P.$$

$$\begin{array}{l}\textbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \land \mathit{inv}(b(v_0))\} \bullet \\ \quad \left(\begin{array}{l}c.e(b(v_0)) \rightarrow \mathit{Skip}; \\ (\textbf{vres}\ x : \mathit{BINDING} \bullet A(x))(b)\end{array}\right)\end{array}$$

[Lemma K.1]

$$= P.$$

$$\begin{array}{l}\textbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \land \mathit{inv}(b(v_0))\} \bullet \\ \quad \left(\begin{array}{l}\textbf{var}\ x : \mathit{BINDING} \bullet \\ \quad c.e(b(v_0)) \rightarrow \mathit{Skip}; \\ \quad A(b)\end{array}\right)\end{array}$$

[Law 34]

$$= P.$$

$$\begin{array}{l}\textbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \land \mathit{inv}(b(v_0))\} \bullet \\ \quad \left(\begin{array}{l}\textbf{var}\ x : \mathit{BINDING} \bullet \\ \quad c.e(b(v_0)) \rightarrow A(b)\end{array}\right)\end{array}$$

[Law 6]

provided

$[x \notin FV(c.e(b(v_0)) \rightarrow A(b))]$

$$= P.$$

$$\begin{array}{l}\textbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \land \mathit{inv}(b(v_0))\} \bullet \\ \quad c.e(b(v_0)) \rightarrow A(b)\end{array}$$

[Law 26(b is the only component of S)]

$$= P_S.(c.e(b(v_0)) \rightarrow A)$$

□

K.6 Output Communications

Theorem K.6

$$P_S.(c!e(b(v_0))) \to A)$$
$$=$$
$$\Omega(P_S.(c.e(b(v_0)) \to A))$$

Proof.

$$\Omega(P_S.(c!e(b(v_0)) \to A))$$

[Ω]

$$= P.\quad \textbf{var } b : \{x : BINDING \mid b(v_0) \in T_0 \wedge inv(b(v_0))\} \bullet \left(\begin{array}{l} (\Omega_A(c!e(b(v_0)) \to A); \\ terminate \to Skip) \\ \llbracket \emptyset \mid MEM_I \mid \{b\} \rrbracket \\ Memory(b) \end{array}\right) \setminus MEM_I$$

[Ω_A]

$$= P.\quad \textbf{var } b : \{x : BINDING \mid b(v_0) \in T_0 \wedge inv(b(v_0))\} \bullet \left(\begin{array}{l} \Omega_A(c.e(b(v_0)) \to A); \\ terminate \to Skip \\ \llbracket \emptyset \mid MEM_I \mid \{b\} \rrbracket \\ Memory(b) \end{array}\right) \setminus MEM_I$$

[Theorem K.5]

$$=$$
$$P_S.(c.e(b(v_0)) \to A)$$

□

K.7 Guard

Theorem K.7

$$P_S(g(b(v_0)) \mathbin{\&} A)$$
$$=$$
$$\Omega(P_S.(g(b(v_0)) \mathbin{\&} A))$$

Proof. Inductive Hypothesis for any state S.

$$
\begin{array}{l}
(\mathbf{vres}\ x : \mathit{BINDING} \bullet A(x))(b) \\
= \\
\left(\begin{array}{l}
(\Omega_A(A);\ \mathit{terminate} \rightarrow \mathit{Skip}) \\
[\![\emptyset \mid \mathit{MEM}_I \mid \{b\}]\!] \\
\mathit{Memory}(b)
\end{array}\right) \setminus \mathit{MEM}_I
\end{array}
$$

□

Proof.

$$\Omega(P_S.(g(b(v_0))\ \&\ A))$$

[Ω]

$$
\begin{array}{l}
= P. \\
\quad \mathbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \wedge \mathit{inv}(b(v_0))\} \bullet \\
\qquad \left(\begin{array}{l}
(\Omega_A(g(b(v_0))\ \&\ A);\ \mathit{terminate} \rightarrow \mathit{Skip}) \\
[\![\emptyset \mid \mathit{MEM}_I \mid \{b\}]\!] \\
\mathit{Memory}(b)
\end{array}\right) \\
\qquad \setminus \mathit{MEM}_I
\end{array}
$$

[Ω_A]

$$
\begin{array}{l}
= P. \\
\quad \mathbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \wedge \mathit{inv}(b(v_0))\} \bullet \\
\qquad \left(\begin{array}{l}
\left(\begin{array}{l}
\mathit{get}.v_0?vv_0 \rightarrow g(vv_0)\ \&\ \Omega_A(A); \\
\mathit{terminate} \rightarrow \mathit{Skip})
\end{array}\right) \\
[\![\emptyset \mid \mathit{MEM}_I \mid \{b\}]\!] \\
\ \mathit{Memory}(b)
\end{array}\right) \\
\qquad \setminus \mathit{MEM}_I
\end{array}
$$

[Lemma K.2]

$$
\begin{array}{l}
= P. \\
\quad \mathbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \wedge \mathit{inv}(b(v_0))\} \bullet \\
\qquad \left(\left(\begin{array}{l}
\left(\begin{array}{l}
\mathit{get}.v_0?vv_0 \rightarrow g(vv_0)\ \&\ \Omega_A(A); \\
\mathit{terminate} \rightarrow \mathit{Skip})
\end{array}\right) \\
[\![\emptyset \mid \mathit{MEM}_I \mid \{b\}]\!] \\
\left(\begin{array}{l}
(\Box\ \mathit{get}.n!b(n) \rightarrow \mathit{Memory}(b)) \\
\Box\ \left(\ \Box\ \mathit{set}.n?nv \rightarrow \mathit{Memory}(b \oplus \{n \mapsto nv\})\ \right) \\
\Box\ \mathit{terminate} \rightarrow \mathit{Skip};
\end{array}\right)
\end{array}\right)\right) \\
\qquad \setminus \mathit{MEM}_I
\end{array}
$$

[Law 10]

provided

$\{terminate\} \subseteq MEM_I$

$\{get, set\} \subseteq MEM_I$

$\{set, terminate\} \in MEM_I$

$\{set, terminate\} \notin \{get\}$

$= P.$

$$\mathbf{var}\ b : \{x : BINDING \mid b(v_0) \in T_0 \land inv(b(v_0))\} \bullet \left(\begin{array}{l} \left(\begin{array}{l} get.v_0?vv_0 \to g(vv_0)\ \&\ \Omega_A(A); \\ terminate \to Skip) \end{array}\right) \\ \llbracket \emptyset \mid MEM_I \mid \{b\} \rrbracket \\ (\ get.v_0!b(v_0) \to Memory(b)\ ) \end{array}\right) \setminus MEM_I$$

[Law 24]

provided

$\{get\} \subseteq MEM_I$

$x \notin FV(Memory(b))$

$= P.$

$$\mathbf{var}\ b : \{x : BINDING \mid b(v_0) \in T_0 \land inv(b(v_0))\} \bullet \left(\begin{array}{l} g(b(v_0))\ \&\ \Omega_A(A); \\ terminate \to Skip) \\ \llbracket \emptyset \mid MEM_I \mid \{b\} \rrbracket \\ Memory(b) \end{array}\right) \setminus MEM_I$$

[Law 52]

provided

$initials(Memory(b)) \subseteq MEM_I$

$= P.$

$$\mathbf{var}\ b : \{x : BINDING \mid b(v_0) \in T_0 \land inv(b(v_0))\} \bullet \left(\begin{array}{l} g(b(v_0))\ \& \\ \qquad \left(\begin{array}{l} \Omega_A(A);\ terminate \to Skip \\ \llbracket \emptyset \mid MEM_I \mid \{b\} \rrbracket \\ Memory(b) \end{array}\right) \end{array}\right) \setminus MEM_I$$

[Law 8]

$= P.$

$$\mathbf{var}\ b : \{x : BINDING \mid b(v_0) \in T_0 \land inv(b(v_0))\} \bullet \left(\begin{array}{l} g(b(v_0))\ \&\\ \quad \left(Skip; \left(\begin{array}{l}(\Omega_A(A);\ terminate \rightarrow Skip\\ [\![\emptyset \mid MEM_I \mid \{b\}]\!]\\ Memory(b))\end{array}\right)\right)\end{array}\right) \setminus MEM_I$$

[Law 32]

$= P.$

$$\mathbf{var}\ b : \{x : BINDING \mid b(v_0) \in T_0 \land inv(b(v_0))\} \bullet \left(\begin{array}{l} g(b(v_0))\ \&\ Skip;\\ \left(\begin{array}{l}\Omega_A(A);\ terminate \rightarrow Skip\\ [\![\emptyset \mid MEM_I \mid \{b\}]\!]\\ Memory(b)\end{array}\right)\end{array}\right) \setminus MEM_I$$

[Law 31]

$= P.$

$$\begin{array}{l}\mathbf{var}\ b : \{x : BINDING \mid b(v_0) \in T_0 \land inv(b(v_0))\} \bullet\\ \quad g(b(v_0))\ \&\ Skip \setminus I\_MEM;\\ \quad \left(\begin{array}{l}\Omega_A(A);\ terminate \rightarrow Skip\\ [\![\emptyset \mid MEM_I \mid \{b\}]\!]\\ Memory(b)\end{array}\right) \setminus MEM_I\end{array}$$

[Law 15]

provided

$[MEM_I \cap usedC(g(b(v_0))\ \&\ Skip) = \emptyset)]$

$= P.$

$$\begin{array}{l}\mathbf{var}\ b : \{x : BINDING \mid b(v_0) \in T_0 \land inv(b(v_0))\} \bullet\\ \quad g(b(v_0))\ \&\ Skip;\\ \quad \left(\begin{array}{l}\Omega_A(A);\ terminate \rightarrow Skip\\ [\![\emptyset \mid MEM_I \mid \{b\}]\!]\\ Memory(b)\end{array}\right) \setminus MEM_I\end{array}$$

[IH]

$= P.$

$$\mathbf{var}\ b : \{x : BINDING \mid b(v_0) \in T_0 \land inv(b(v_0))\} \bullet \left(\begin{array}{l} g(b(v_0))\ \&\ Skip;\\ (\mathbf{vres}\ x : BINDING \bullet A(x))(b)\end{array}\right)$$

[Lemma K.1]

$= P.$

$$\begin{array}{l}\textbf{var}\ b : \{x : BINDING \mid b(v_0) \in T_0 \land inv(b(v_0))\} \bullet \\ \quad \left(\begin{array}{l}\textbf{var}\ x : BINDING \bullet \\ \quad g(b(v_0))\ \&\ Skip; \\ \quad A(b)\end{array}\right)\end{array}$$

[Law 32]

$= P.$

$$\begin{array}{l}\textbf{var}\ b : \{x : BINDING \mid b(v_0) \in T_0 \land inv(b(v_0))\} \bullet \\ \quad \left(\begin{array}{l}\textbf{var}\ x : BINDING \bullet \\ \quad g(b(v_0))\ \& \\ \quad Skip; A(b)\end{array}\right)\end{array}$$

[Law 8]

$= P.$

$$\begin{array}{l}\textbf{var}\ b : \{x : BINDING \mid b(v_0) \in T_0 \land inv(b(v_0))\} \bullet \\ \quad \left(\begin{array}{l}(\textbf{var}\ x : BINDING \bullet \\ \quad g(b(v_0))\ \&\ A(b)\end{array}\right)\end{array}$$

[Law 6]

provided

$[x \notin FV(g(b(v_0))\ \&\ A(b))]$

$= P.$

$$\begin{array}{l}\textbf{var}\ b : \{x : BINDING \mid b(v_0) \in T_0 \land inv(b(v_0))\} \bullet \\ \quad g(b(v_0))\ \&\ A(b)\end{array}$$

[Law 26(b is the only component of S)]

$=$

$P_S.(g(b(v_0))\ \&\ A) = P_S.(g(v_0)\ \&\ A)$

□

K.8 Input

Theorem K.8

$P_S(c?x : P(x, b(v_0)) \to A)$

$=$

$\Omega(Ps(c?x : P(x, b(v_0)) \to A))$

Proof. Inductive Hypothesis for any state S.

$$
\begin{array}{l}
(\mathbf{vres}\ x : \mathit{BINDING} \bullet A(x))(b) \\
= \\
\left(\begin{array}{l}
(\Omega_A(A);\ \mathit{terminate} \to \mathit{Skip}) \\
[\![\emptyset \mid \mathit{MEM}_I \mid \{b\}]\!] \\
\mathit{Memory}(b)
\end{array}\right) \setminus \mathit{MEM}_I
\end{array}
$$

□

Proof.

$$\Omega(P_S.(c?x : P(x, b(v_0)) \to A))$$

[Ω]

$$
\begin{array}{l}
= P. \\
\quad \mathbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \land \mathit{inv}(b(v_0))\} \bullet \\
\quad \left(\begin{array}{l}
\Omega_A(c?x : P(x, b(v_0)) \to A); \\
\mathit{terminate} \to \mathit{Skip} \\
[\![\emptyset \mid \mathit{MEM}_I \mid \{b\}]\!] \\
\mathit{Memory}(b)
\end{array}\right) \setminus \mathit{MEM}_I
\end{array}
$$

[Ω_A]

$$
\begin{array}{l}
= P. \\
\quad \mathbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \land \mathit{inv}(b(v_0))\} \bullet \\
\quad \left(\begin{array}{l}
\left(\begin{array}{l}
\mathit{get}.v_0?vv_0 \to c?x : P(x, vv_0) \to \Omega_A(A); \\
\mathit{terminate} \to \mathit{Skip}
\end{array}\right) \\
[\![\emptyset \mid \mathit{MEM}_I \mid \{b\}]\!] \\
\mathit{Memory}(b)
\end{array}\right) \setminus \mathit{MEM}_I
\end{array}
$$

[Lemma K.2]

$$
\begin{array}{l}
= P. \\
\quad \mathbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \land \mathit{inv}(b(v_0))\} \bullet \\
\quad \left(\left(\begin{array}{l}
\left(\begin{array}{l}
\mathit{get}.v_0?vv_0 \to c?x : P(x, vv_0) \to \Omega_A(A); \\
\mathit{terminate} \to \mathit{Skip}
\end{array}\right) \\
[\![\emptyset \mid \mathit{MEM}_I \mid \{b\}]\!] \\
\left(\begin{array}{l}
(\Box\ \mathit{get}.n!b(n) \to \mathit{Memory}(b)) \\
\Box\ (\ \Box\ \mathit{set}.n?nv \to \mathit{Memory}(b \oplus \{n \mapsto nv\})\ ) \\
\Box\ \mathit{terminate} \to \mathit{Skip};
\end{array}\right)
\end{array}\right)\right) \setminus \mathit{MEM}_I
\end{array}
$$

[Law 10]

provided

$\{terminate\} \subseteq MEM_I$

$\{get, set\} \subseteq MEM_I$

$\{set, terminate\} \in MEM_I$

$\{set, terminate\} \notin \{get\}$

$= P.$

$$\textbf{var}\ b : \{x : BINDING \mid b(v_0) \in T_0 \wedge inv(b(v_0))\} \bullet \left(\begin{array}{l} \left(\begin{array}{l} get.v_0?vv_0 \to c?x : P(x, vv_0) \to \Omega_A(A); \\ terminate \to Skip \end{array}\right) \\ [\![\emptyset \mid MEM_I \mid \{b\}]\!] \\ \left(\ get.v_0!b(v_0) \to Memory(b)\ \right) \end{array}\right) \setminus MEM_I$$

[Law 24]

provided

$\{get\} \subseteq MEM_I$

$x \notin FV(Memory(b))$

$= P.$

$$\textbf{var}\ b : \{x : BINDING \mid b(v_0) \in T_0 \wedge inv(b(v_0))\} \bullet \left(\begin{array}{l} c?x : P(x, b(v_0)) \to \Omega_A(A); \\ terminate \to Skip \\ [\![\emptyset \mid MEM_I \mid \{b\}]\!] \\ Memory(b) \end{array}\right) \setminus MEM_I$$

[Law 33]

provided

$[c \notin MEM_I]$

$[x \notin usedV(Memory(b))]$

$[initials(Memory(b)) \subseteq MEM_I]$

$[Memory(b)$ is deterministic$]$

$= P.$

$$\textbf{var}\ b : \{x : BINDING \mid b(v_0) \in T_0 \wedge inv(b(v_0))\} \bullet \left(\begin{array}{l} c?x : P(x, b(v_0)) \to \\ \quad \left(\begin{array}{l} \Omega_A(A);\ terminate \to Skip) \\ [\![\emptyset \mid MEM_I \mid \{b\}]\!] \\ Memory(b) \end{array}\right) \end{array}\right) \setminus MEM_I$$

[Law 34]

$= P.$

$$\mathbf{var}\ b : \{x : BINDING \mid b(v_0) \in T_0 \land inv(b(v_0))\} \bullet \left(\begin{array}{l} c?x : P(x, b(v_0)) \rightarrow Skip; \\ \left(\begin{array}{l} \Omega_A(A);\ terminate \rightarrow Skip) \\ [\![\emptyset \mid MEM_I \mid \{b\}]\!] \\ Memory(b) \end{array} \right) \end{array} \right) \setminus MEM_I$$

[Law 31]

$= P.$

$$\mathbf{var}\ b : \{x : BINDING \mid b(v_0) \in T_0 \land inv(b(v_0))\} \bullet \begin{array}{l} (c?x : P(x, b(v_0)) \rightarrow Skip) \setminus MEM_I; \\ \left(\begin{array}{l} \Omega_A(A);\ terminate \rightarrow Skip) \\ [\![\emptyset \mid MEM_I \mid \{b\}]\!] \\ Memory(b) \end{array} \right) \setminus MEM_I \end{array}$$

[Law 15]

provided

$[MEM_I \cap usedC(c?x : P(x, b(v_0)) \rightarrow Skip) = \emptyset)]$

$= P.$

$$\mathbf{var}\ b : \{x : BINDING \mid b(v_0) \in T_0 \land inv(b(v_0))\} \bullet \begin{array}{l} c?x : P(x, b(v_0))) \rightarrow Skip; \\ \left(\begin{array}{l} \Omega_A(A);\ terminate \rightarrow Skip) \\ [\![\emptyset \mid MEM_I \mid \{b\}]\!] \\ Memory(b) \end{array} \right) \setminus MEM_I \end{array}$$

[IH]

$= P.$

$$\mathbf{var}\ b : \{x : BINDING \mid b(v_0) \in T_0 \land inv(b(v_0))\} \bullet \left(\begin{array}{l} c?x : P(x, b(v_0)) \rightarrow Skip; \\ (\mathbf{vres}\ x : BINDING \bullet A(x))(b) \end{array} \right)$$

[Lemma K.1]

$= P.$

$$\mathbf{var}\ b : \{x : BINDING \mid b(v_0) \in T_0 \land inv(b(v_0))\} \bullet \left(\begin{array}{l} \mathbf{var}\ x : BINDING \bullet \\ \qquad c?x : P(x, b(v_0)) \rightarrow Skip; \\ \qquad A(b) \end{array} \right)$$

[Law 34]

$= P.$

$$\begin{array}{l} \textbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \wedge \mathit{inv}(b(v_0))\} \bullet \\ \quad \left(\begin{array}{l} \textbf{var}\ x : \mathit{BINDING} \bullet \\ \quad c?x : P(x, b(v_0)) \to A(b) \end{array} \right) \end{array}$$

[Law 6]

provided

$[x \notin FV(c?x : P(x, b(v_0)) \to A(b))]$

$= P.$

$$\begin{array}{l} \textbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \wedge \mathit{inv}(b(v_0))\} \bullet \\ \quad c?x : P(x, b(v_0)) \to A(b) \end{array}$$

[Law 26(b is the only component of S)]

$=$

$P_S.(c?x : P(x, b(v_0)) \to A)$

□

K.9 Internal Choice

Theorem K.9

$$P_S.(A_1 \sqcap A_2)$$
$$=$$
$$\Omega(P_S.(A_1 \sqcap A_2))$$

Proof. Inductive Hypothesis: for any state S_1 and S_2

$$(\textbf{vres}\ x : \mathit{BINDING} \bullet A_1(x))(b)$$
$$=$$
$$\left(\begin{array}{l} (\Omega_A(A_1);\ \mathit{terminate} \to \mathit{Skip}) \\ [\![\emptyset \mid \mathit{MEM}_I \mid \{b\}]\!] \\ \mathit{Memory}(b) \end{array} \right) \setminus \mathit{MEM}_I$$

and

$$(\textbf{vres}\ x : \mathit{BINDING} \bullet A_2(x))(b) =$$
$$\left(\begin{array}{l} (\Omega_A(A_2);\ \mathit{terminate} \to \mathit{Skip}) \\ [\![\emptyset \mid \mathit{MEM}_I \mid \{b\}]\!] \\ \mathit{Memory}(b) \end{array} \right) \setminus \mathit{MEM}_I$$

Proof.

$$\Omega(P_S.(A1 \sqcap A2))$$

[Ω]

$$= P.$$

$$\mathbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \land \mathit{inv}(b(v_0))\} \bullet \left(\begin{array}{l} (\Omega_A(A_1 \sqcap A_2);\ \mathit{terminate} \rightarrow \mathit{Skip}) \\ [\![\emptyset \mid \mathit{MEM}_I \mid \{b\}]\!] \\ \mathit{Memory}(b) \end{array}\right) \setminus \mathit{MEM}_I$$

[Ω_A]

$$= P.$$

$$\mathbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \land \mathit{inv}(b(v_0))\} \bullet \left(\begin{array}{l} (\Omega_A(A_1) \sqcap \Omega_A(A_2);\ \mathit{terminate} \rightarrow \mathit{Skip}) \\ [\![\emptyset \mid \mathit{MEM}_I \mid \{b\}]\!] \\ \mathit{Memory}(b) \end{array}\right) \setminus \mathit{MEM}_I$$

[Law 43]

$$= P.$$

$$\mathbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \land \mathit{inv}(b(v_0))\} \bullet \left(\begin{array}{l} \left(\begin{array}{l} ((\Omega_A(A1);\mathit{terminate} \rightarrow \mathit{Skip}) \\ \sqcap\ (\Omega_A(A2);\mathit{terminate} \rightarrow \mathit{Skip})) \end{array}\right) \\ [\![\emptyset \mid \mathit{MEM}_I \mid \{b\}]\!] \\ \mathit{Memory}(b)) \end{array}\right) \setminus \mathit{MEM}_I$$

[Law 42]

$$= P.$$

$$\mathbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \land \mathit{inv}(b(v_0))\} \bullet \left(\begin{array}{l} (\Omega_A(A1);\mathit{terminate} \rightarrow \mathit{Skip} \\ [\![\emptyset \mid \mathit{MEM}_I \mid \{b\}]\!] \\ \mathit{Memory}(b)) \\ \sqcap \\ (\Omega_A(A2);\mathit{terminate} \rightarrow \mathit{Skip} \\ [\![\emptyset \mid \mathit{MEM}_I \mid \{b\}]\!] \\ \mathit{Memory}(b)) \end{array}\right) \setminus \mathit{MEM}_I$$

[Law 53]

$$= P.$$

$$\begin{array}{l}\textbf{var}\ b : \{x : BINDING \mid b(v_0) \in T_0 \land inv(b(v_0))\} \bullet \\ \left(\begin{array}{l}\left(\begin{array}{l}(\Omega_A(A1); terminate \to Skip \\ \llbracket \emptyset \mid MEM_I \mid \{b\} \rrbracket \\ Memory(b))\end{array}\right) \setminus MEM_I \\ \sqcap \\ \left(\begin{array}{l}(\Omega_A(A2); terminate \to Skip \\ \llbracket \emptyset \mid MEM_I \mid \{b\} \rrbracket \\ Memory(b))\end{array}\right) \setminus MEM_I\end{array}\right)\end{array}$$

[IH]

$= P.$

$$\begin{array}{l}\textbf{var}\ b : \{x : BINDING \mid b(v_0) \in T_0 \land inv(b(v_0))\} \bullet \\ \left(\begin{array}{l}(\textbf{vres}\ x : BINDING \bullet A_1(x))(b) \sqcap \\ (\textbf{vres}\ x : BINDING \bullet A_2(x))(b)\end{array}\right)\end{array}$$

[Semantics]

$= P.$

$$\begin{array}{l}\textbf{var}\ b : \{x : BINDING \mid b(v_0) \in T_0 \land inv(b(v_0))\} \bullet \\ \left(\begin{array}{l}(\textbf{var}\ x : BINDING \bullet x := b;\ A_1(x);\ b := x) \sqcap \\ (\textbf{var}\ x : BINDING \bullet x := b;\ A_2(x);\ b := x)\end{array}\right)\end{array}$$

[Law 49]

$= P.$

$$\begin{array}{l}\textbf{var}\ b : \{x : BINDING \mid b(v_0) \in T_0 \land inv(b(v_0))\} \bullet \\ \left(\begin{array}{l}(\textbf{var}\ x : BINDING \bullet x := b;\ A_1(x);\ b := x) \sqcap \\ (\textbf{var}\ y : BINDING \bullet y := b;\ A_2(y);\ b := y)\end{array}\right)\end{array}$$

[Law 47]

$= P.$

$$\begin{array}{l}\textbf{var}\ b : \{x : BINDING \mid b(v_0) \in T_0 \land inv(b(v_0))\} \bullet \\ \left(\begin{array}{l}(\textbf{var}\ x : BINDING \bullet x := b;\ A_1(x);\ b := x) \sqcap \\ (\textbf{var}\ y : BINDING \bullet A_2(b);\ b := b)\end{array}\right)\end{array}$$

[Laws 48 and 8]

$= P.$

$$\begin{array}{l}\textbf{var}\ b : \{x : BINDING \mid b(v_0) \in T_0 \land inv(b(v_0))\} \bullet \\ \left(\begin{array}{l}(\textbf{var}\ x : BINDING \bullet x := b;\ A_1(x);\ b := x) \sqcap \\ (\textbf{var}\ y : BINDING \bullet A_2(b))\end{array}\right)\end{array}$$

[Law 47]

$= P.$

$$\textbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \wedge \mathit{inv}(b(v_0))\} \bullet \left(\begin{array}{l} (\textbf{var}\ x : \mathit{BINDING} \bullet A_1(b);\ b := b) \sqcap \\ (\textbf{var}\ y : \mathit{BINDING} \bullet A_2(b)) \end{array} \right)$$

[Laws 48 and 8]

$= P.$

$$\textbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \wedge \mathit{inv}(b(v_0))\} \bullet \left(\begin{array}{l} (\textbf{var}\ x : \mathit{BINDING} \bullet A_1(b)) \sqcap \\ (\textbf{var}\ y : \mathit{BINDING} \bullet A_2(b)) \end{array} \right)$$

[Law 6]

provided

$[x \notin FV(A_2(b))]$

$= P.$

$$\textbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \wedge \mathit{inv}(b(v_0))\} \bullet \left(\left(\textbf{var}\ x : \mathit{BINDING} \bullet A_1(b) \right) \sqcap A_2(b) \right)$$

[Law 6]

provided

$[x \notin FV(A_1(b))]$

$= P.$

$$\textbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \wedge \mathit{inv}(b(v_0))\} \bullet \left(A_1(b) \sqcap A_2(b) \right)$$

[Law 26(b is the only component of S)]

$=$

$P_S.(A_1 \sqcap A_2)$

$\square$

K.10 External Choice

Theorem K.10

$P_S.(A_1 \square A_2)$

$=$

$\Omega(P_S.(A_1 \square A_2))$

Proof. Inductive Hypothesis: for any state S_1 and S_2

$$\begin{array}{l}
(\mathbf{vres}\ x : \mathit{BINDING} \bullet A_1(x))(b) \\
= \\
\left(\begin{array}{l}(\Omega_A(A_1);\ \mathit{terminate} \rightarrow \mathit{Skip}) \\ \lbrack\!\lbrack \emptyset \mid \mathit{MEM}_I \mid \{b\} \rbrack\!\rbrack \\ \mathit{Memory}(b)\end{array}\right) \setminus \mathit{MEM}_I \\
\text{and} \\
(\mathbf{vres}\ x : \mathit{BINDING} \bullet A_2(x))(b) = \\
\left(\begin{array}{l}(\Omega_A(A_2);\ \mathit{terminate} \rightarrow \mathit{Skip}) \\ \lbrack\!\lbrack \emptyset \mid \mathit{MEM}_I \mid \{b\} \rbrack\!\rbrack \\ \mathit{Memory}(b)\end{array}\right) \setminus \mathit{MEM}_I
\end{array}$$

Proof.

$$\begin{array}{ll}
\Omega(P_S.(A1 \Box A2)) & [\Omega] \\
= P. & \\
\quad \mathbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \land \mathit{inv}(b(v_0))\} \bullet & \\
\quad\quad \left(\begin{array}{l}(\Omega_A(A_1 \Box A_2);\mathit{terminate} \rightarrow \mathit{Skip}) \\ \lbrack\!\lbrack \emptyset \mid \mathit{MEM}_I \mid \{b\} \rbrack\!\rbrack \\ \mathit{Memory}(b)\end{array}\right) \setminus \mathit{MEM}_I & \\
 & [\Omega_A] \\
= P. & \\
\quad \mathbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \land \mathit{inv}(b(v_0))\} \bullet & \\
\quad\quad \left(\begin{array}{l}\left(\begin{array}{l}\mathit{get}.v_0?vv_0 \rightarrow \Omega'_A(A_1) \Box \Omega'_A(A_2); \\ \mathit{terminate} \rightarrow \mathit{Skip}\end{array}\right) \\ \lbrack\!\lbrack \emptyset \mid \mathit{MEM}_I \mid \{b\} \rbrack\!\rbrack \\ \mathit{Memory}(b)\end{array}\right) & \\
\quad\quad \setminus \mathit{MEM}_I & \\
 & [\text{Lemma K.2}] \\
= P. & \\
\quad \mathbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \land \mathit{inv}(b(v_0))\} \bullet & \\
\quad\quad \left(\left(\begin{array}{l}\left(\begin{array}{l}\mathit{get}.v_0?vv_0 \rightarrow \Omega'_A(A_1) \Box \Omega'_A(A_2); \\ \mathit{terminate} \rightarrow \mathit{Skip}\end{array}\right) \\ \lbrack\!\lbrack \emptyset \mid \mathit{MEM}_I \mid \{b\} \rbrack\!\rbrack \\ \left(\begin{array}{l}(\Box\, \mathit{get}.n!b(n) \rightarrow \mathit{Memory}(b)) \\ \Box \left(\Box\, \mathit{set}.n?nv \rightarrow \mathit{Memory}(b \oplus \{n \mapsto nv\}) \right) \\ \Box\ \mathit{terminate} \rightarrow \mathit{Skip};\end{array}\right)\end{array}\right)\right) & \\
\quad\quad \setminus \mathit{MEM}_I &
\end{array}$$

[Law 10]

provided

$\{terminate\} \subseteq MEM_I$

$\{get, set\} \subseteq MEM_I$

$\{set, terminate\} \in MEM_I$

$\{set, terminate\} \notin \{get\}$

$= P.$

$$\mathbf{var}\ b : \{x : BINDING \mid b(v_0) \in T_0 \wedge inv(b(v_0))\} \bullet \left(\begin{array}{l} \left(\begin{array}{l} get.v_0?vv_0 \to \Omega'_A(A_1) \Box \Omega'_A(A_2); \\ terminate \to Skip \end{array}\right) \\ [\![\emptyset \mid MEM_I \mid \{b\}]\!] \\ \left(get.v_0!b(v_0) \to Memory(b) \right) \end{array}\right) \setminus MEM_I$$

[Law 24]

provided

$\{get\} \subseteq MEM_I$

$x \notin FV(Memory(b))$

$= P.$

$$\mathbf{var}\ b : \{x : BINDING \mid b(v_0) \in T_0 \wedge inv(b(v_0))\} \bullet \left(\begin{array}{l} \Omega'_A(A_1) \Box \Omega'_A(A_2); terminate \to Skip \\ [\![\emptyset \mid MEM_I \mid \{b\}]\!] \\ Memory(b) \end{array}\right) \setminus MEM_I$$

[Law 37]

$= P.$

$$\mathbf{var}\ b : \{x : BINDING \mid b(v_0) \in T_0 \wedge inv(b(v_0))\} \bullet \left(\begin{array}{l} (true \ \& \ \Omega'_A(A_1)) \Box \ (true \ \& \ \Omega'_A(A_2)); \\ terminate \to Skip \\ [\![\emptyset \mid MEM_I \mid \{b\}]\!] \\ Memory(b) \end{array}\right) \setminus MEM_I$$

[Law 19]

$= P.$

$$\begin{array}{l}\textbf{var}\ b : \{x : BINDING \mid b(v_0) \in T_0 \wedge inv(b(v_0))\} \bullet \\ \quad \left(\begin{array}{l}\left(\begin{array}{l}(true\ \&\ \Omega'_A(A_1));\ terminate \rightarrow Skip \\ \Box\ (true\ \&\ \Omega'_A(A_2));\ terminate \rightarrow Skip\end{array}\right) \\ \lbrack\!\lbrack \emptyset \mid MEM_I \mid \{b\} \rbrack\!\rbrack \\ Memory(b)\end{array}\right) \\ \quad \setminus MEM_I\end{array}$$

[Law 37]

$= P.$

$$\begin{array}{l}\textbf{var}\ b : \{x : BINDING \mid b(v_0) \in T_0 \wedge inv(b(v_0))\} \bullet \\ \quad \left(\begin{array}{l}\left(\begin{array}{l}\Omega'_A(A_1);\ terminate \rightarrow Skip \\ \Box\ \Omega'_A(A_2);\ terminate \rightarrow Skip\end{array}\right) \\ \lbrack\!\lbrack \emptyset \mid MEM_I \mid \{b\} \rbrack\!\rbrack \\ Memory(b)\end{array}\right) \\ \quad \setminus MEM_I\end{array}$$

In our strategy, the External Choices are only among prefixed actions. So, in this case, we can use the Law 17

provided

$initials(Memory(b)) \subseteq MEM_I$

Memory(b) is Deterministic

$= P.$

$$\begin{array}{l}\textbf{var}\ b : \{x : BINDING \mid b(v_0) \in T_0 \wedge inv(b(v_0))\} \bullet \\ \quad \left(\begin{array}{l}(\Omega'_A(A1); terminate \rightarrow Skip \\ \lbrack\!\lbrack \emptyset \mid MEM_I \mid \{b\} \rbrack\!\rbrack \\ Memory(b)) \\ \Box \\ (\Omega'_A(A2); terminate \rightarrow Skip \\ \lbrack\!\lbrack \emptyset \mid MEM_I \mid \{b\} \rbrack\!\rbrack \\ Memory(b))\end{array}\right) \setminus MEM_I\end{array}$$

[Law 20]

This is valid because we force the structure of the actions in external choices to be prefixed actions and because $\Omega'_A(A1)$ does not include any events from MEM_I

$= P.$

$$\mathbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \land \mathit{inv}(b(v_0))\} \bullet$$
$$\left(\begin{array}{l}\left(\begin{array}{l}(\Omega'_A(A1); \mathit{terminate} \rightarrow \mathit{Skip}\\ \lbrack\!\lbrack \emptyset \mid \mathit{MEM}_I \mid \{b\} \rbrack\!\rbrack\\ \mathit{Memory}(b))\end{array}\right) \setminus \mathit{MEM}_I\\ \Box\\ \left(\begin{array}{l}(\Omega'_A(A2); \mathit{terminate} \rightarrow \mathit{Skip}\\ \lbrack\!\lbrack \emptyset \mid \mathit{MEM}_I \mid \{b\} \rbrack\!\rbrack\\ \mathit{Memory}(b))\end{array}\right) \setminus \mathit{MEM}_I\end{array}\right)$$

[IH]

$= P.$

$$\mathbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \land \mathit{inv}(b(v_0))\} \bullet$$
$$\left(\begin{array}{l}(\mathbf{vres}\ x : \mathit{BINDING} \bullet A_1(x))(b)\ \Box\\ (\mathbf{vres}\ x : \mathit{BINDING} \bullet A_2(x))(b)\end{array}\right)$$

[Semantics]

$= P.$

$$\mathbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \land \mathit{inv}(b(v_0))\} \bullet$$
$$\left(\begin{array}{l}(\mathbf{var}\ x : \mathit{BINDING} \bullet x := b;\ A_1(x);\ b := x)\ \Box\\ (\mathbf{var}\ x : \mathit{BINDING} \bullet x := b;\ A_2(x);\ b := x)\end{array}\right)$$

[Law 49]

$= P.$

$$\mathbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \land \mathit{inv}(b(v_0))\} \bullet$$
$$\left(\begin{array}{l}(\mathbf{var}\ x : \mathit{BINDING} \bullet x := b;\ A_1(x);\ b := x)\ \Box\\ (\mathbf{var}\ y : \mathit{BINDING} \bullet y := b;\ A_2(y);\ b := y)\end{array}\right)$$

[Law 47]

$= P.$

$$\mathbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \land \mathit{inv}(b(v_0))\} \bullet$$
$$\left(\begin{array}{l}(\mathbf{var}\ x : \mathit{BINDING} \bullet x := b;\ A_1(x);\ b := x)\ \Box\\ (\mathbf{var}\ y : \mathit{BINDING} \bullet A_2(b);\ b := b)\end{array}\right)$$

[Laws 48 and 8]

$= P.$

$$\mathbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \land \mathit{inv}(b(v_0))\} \bullet$$
$$\left(\begin{array}{l}(\mathbf{var}\ x : \mathit{BINDING} \bullet x := b;\ A_1(x);\ b := x)\ \Box\\ (\mathbf{var}\ y : \mathit{BINDING} \bullet A_2(b))\end{array}\right)$$

[Law 47]

$= P.$

$$\begin{array}{l}\textbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \wedge \mathit{inv}(b(v_0))\} \bullet \\ \quad \left(\begin{array}{l}(\textbf{var}\ x : \mathit{BINDING} \bullet A_1(b);\ b := b) \Box \\ (\textbf{var}\ y : \mathit{BINDING} \bullet A_2(b))\end{array}\right)\end{array}$$

[Laws 48 and 8]

$= P.$

$$\begin{array}{l}\textbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \wedge \mathit{inv}(b(v_0))\} \bullet \\ \quad \left(\begin{array}{l}(\textbf{var}\ x : \mathit{BINDING} \bullet A_1(b)) \Box \\ (\textbf{var}\ y : \mathit{BINDING} \bullet A_2(b))\end{array}\right)\end{array}$$

[Law 6]

provided

$[x \notin FV(A_2(b))]$

$= P.$

$$\begin{array}{l}\textbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \wedge \mathit{inv}(b(v_0))\} \bullet \\ \quad \left(\left(\textbf{var}\ x : \mathit{BINDING} \bullet A_1(b) \right) \Box A_2(b) \right)\end{array}$$

[Law 6]

provided

$[x \notin FV(A_1(b))]$

$= P.$

$$\begin{array}{l}\textbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \wedge \mathit{inv}(b(v_0))\} \bullet \\ \quad \left(A_1(b) \Box A_2(b) \right)\end{array}$$

[Law 26(b is the only component of S)]

$=$

$P_S.(A_1 \Box\ A_2)$

$\Box$

K.11 Hiding

Theorem K.11

$P_S.(A \setminus cs)$

$=$

$\Omega(P_S.(A \setminus cs))$

Proof. Inductive Hypothesis for any state S.

$$
\begin{array}{l}
(\mathbf{vres}\ x : \mathit{BINDING} \bullet A(x))(b) \\
= \\
\left(\begin{array}{l}
(\Omega_A(A); \mathit{terminate} \rightarrow \mathit{Skip}) \\
[\![\emptyset \mid \mathit{MEM}_I \mid \{b\}]\!] \\
\mathit{Memory}(b)
\end{array}\right) \setminus \mathit{MEM}_I
\end{array}
$$

□

Proof.

$$
\begin{array}{ll}
\Omega(P_S.(A \setminus cs)) & \\
& [\Omega] \\
= P. & \\
\quad \mathbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \wedge \mathit{inv}(b(v_0))\} \bullet & \\
\quad\quad \left(\begin{array}{l}
(\Omega(A \setminus cs); \mathit{terminate} \rightarrow \mathit{Skip} \\
[\![\emptyset \mid \mathit{MEM}_I \mid \{b\}]\!] \\
\mathit{Memory}(b)
\end{array}\right) \setminus \mathit{MEM}_I & \\
& [\Omega_A] \\
= P. & \\
\quad \mathbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \wedge \mathit{inv}(b(v_0))\} \bullet & \\
\quad\quad \left(\begin{array}{l}
\Omega_A(A) \setminus cs; \mathit{terminate} \rightarrow \mathit{Skip} \\
[\![\emptyset \mid \mathit{MEM}_I \mid \{b\}]\!] \\
\mathit{Memory}(b)
\end{array}\right) \setminus \mathit{MEM}_I & \\
& [\text{Law } 15] \\
\quad \mathbf{provided} & \\
\quad [cs \cap \mathit{usedC}(\mathit{terminate} \rightarrow \mathit{Skip}) = \emptyset] & \\
= P. & \\
\quad \mathbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \wedge \mathit{inv}(b(v_0))\} \bullet & \\
\quad\quad \left(\begin{array}{l}
(\Omega_A(A)) \setminus cs; (\mathit{terminate} \rightarrow \mathit{Skip}) \setminus cs \\
[\![\emptyset \mid \mathit{MEM}_I \mid \{b\}]\!] \\
\mathit{Memory}(b)
\end{array}\right) \setminus \mathit{MEM}_I & \\
& [\text{Law } 31] \\
= P. &
\end{array}
$$

$$\mathbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \land \mathit{inv}(b(v_0))\} \bullet \left(\begin{array}{l} (\Omega_A(A);\ \mathit{terminate} \to \mathit{Skip}) \setminus cs \\ [\![\emptyset \mid \mathit{MEM}_I \mid \{b\}]\!] \\ \mathit{Memory}(b) \end{array}\right) \setminus \mathit{MEM}_I$$

[Law 15]

provided

$[cs \cap \mathit{usedC}(\mathit{Memory}(b)) = \emptyset]$

$= P.$

$$\mathbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \land \mathit{inv}(b(v_0))\} \bullet \left(\begin{array}{l} (\Omega_A(A);\ \mathit{terminate} \to \mathit{Skip}) \setminus cs \\ [\![\emptyset \mid \mathit{MEM}_I \mid \{b\}]\!] \\ \mathit{Memory}(b) \setminus cs \end{array}\right) \setminus \mathit{MEM}_I$$

[Law 44]

provided

$[\mathit{MEM}_I \cap cs = \emptyset]$

$= P.$

$$\mathbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \land \mathit{inv}(b(v_0))\} \bullet \left(\begin{array}{l} (\Omega_A(A);\ \mathit{terminate} \to \mathit{Skip}) \\ [\![\emptyset \mid \mathit{MEM}_I \mid \{b\}]\!] \\ \mathit{Memory}(b) \end{array}\right) \setminus \mathit{MEM}_I \cup cs$$

[Law 45]

$= P.$

$$\mathbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \land \mathit{inv}(b(v_0))\} \bullet \left(\left(\begin{array}{l} (\Omega_A(A);\ \mathit{terminate} \to \mathit{Skip}) \\ [\![\emptyset \mid \mathit{MEM}_I \mid \{b\}]\!] \\ \mathit{Memory}(b) \end{array}\right) \setminus \mathit{MEM}_I\right) \setminus cs$$

[IH]

$= P.$

$$\mathbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \land \mathit{inv}(b(v_0))\} \bullet \left(\ (\mathbf{vres}\ x : \mathit{BINDING} \bullet A(x))(b)\ \right) \setminus cs$$

[Semantics]

$= P.$

$$\mathbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \land \mathit{inv}(b(v_0))\} \bullet \left(\ \mathbf{var}\ x : \mathit{BINDING} \bullet x := b;\ A(x);\ b := x\ \right) \setminus cs$$

[Law 47]

$= P.$

$$\begin{array}{l}\textbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \wedge \mathit{inv}(b(v_0))\} \bullet \\ \quad \big(\ \textbf{var}\ x : \mathit{BINDING} \bullet A(b);\ b := b\ \big) \setminus cs\end{array}$$

[Laws 48 and 8]

$= P.$

$$\begin{array}{l}\textbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \wedge \mathit{inv}(b(v_0))\} \bullet \\ \quad \big(\ \textbf{var}\ x : \mathit{BINDING} \bullet A(b)\ \big) \setminus cs\end{array}$$

[Law 6]

provided

$[x \notin FV(A(b))]$

$= P.$

$$\begin{array}{l}\textbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \wedge \mathit{inv}(b(v_0))\} \bullet \\ \quad A(b) \setminus cs\end{array}$$

[Law 26(b is the only component of S)]

$=$

$P_S.(A \setminus cs)$

□

K.12 Alternation

Theorem K.12

$$P_S \left(\begin{array}{l} \textbf{if}\ g_0(v_0) \to\ A_0 \\ \quad [\!]\ \dots \\ \quad [\!]\ g_n(v_0) \to\ A_n \\ \textbf{fi} \end{array} \right)$$

$=$

$$\Omega(P_S. \left(\begin{array}{l} \textbf{if}\ g_0(vv_0) \to\ A_0 \\ \quad [\!]\ \dots \\ \quad [\!]\ g_n(vv_0) \to\ A_n \\ \textbf{fi} \end{array} \right))$$

Inductive Hypothesis: for every $i \in \{0, \ldots, n\}$.

$$\begin{array}{l}
(\mathbf{vres}\ x : \mathit{BINDING} \bullet A_i(x))(b) \\
= \\
\left(\begin{array}{l} (\Omega_A(A_i);\ \mathit{terminate} \rightarrow \mathit{Skip}) \\ [\![\emptyset \mid \mathit{MEM}_I \mid \{b\}]\!] \\ \mathit{Memory}(b) \end{array}\right) \setminus \mathit{MEM}_I
\end{array}$$

Proof.

$$\Omega\left(\begin{array}{l} P. \\ \qquad \mathbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \wedge inv(b(v_0))\} \bullet \\ \qquad\qquad \left(\begin{array}{l} \mathbf{if}\ g_0(vv_0) \rightarrow\ A_0 \\ \quad [\!] \ldots \\ \quad [\!]\ g_n(vv_0) \rightarrow\ A_n \\ \mathbf{fi} \end{array}\right) \end{array}\right) \qquad [\Omega]$$

$$\begin{array}{l}
= P. \\
\quad \mathbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \wedge inv(b(v_0))\} \bullet \\
\qquad \left(\begin{array}{l} \Omega_A\left(\begin{array}{l} \mathbf{if}\ g_0(vv_0) \rightarrow\ A_0 \\ \quad [\!] \ldots \\ \quad [\!]\ g_n(vv_0) \rightarrow\ A_n \\ \mathbf{fi} \end{array}\right); \\ \mathit{terminate} \rightarrow \mathit{Skip} \\ [\![\emptyset \mid \mathit{MEM}_I \mid \{b\}]\!] \\ \mathit{Memory}(b) \end{array}\right) \\
\qquad \setminus \mathit{MEM}_I
\end{array}$$

$[\Omega_A]$

$$\begin{array}{l}
= P. \\
\quad \mathbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \wedge inv(b(v_0))\} \bullet \\
\qquad \left(\begin{array}{l} \left(\begin{array}{l} get.v_0?vv_0 \rightarrow \\ \mathbf{if}\ g_0(vv_0) \rightarrow\ \Omega_A(A_0) \\ [\!] \ldots \\ [\!]\ g_n(vv_0) \rightarrow\ \Omega_A(A_n) \\ \mathbf{fi} \end{array}\right); \\ \mathit{terminate} \rightarrow \mathit{Skip} \\ [\![\emptyset \mid \mathit{MEM}_I \mid \{b\}]\!] \\ \mathit{Memory}(b) \end{array}\right) \\
\qquad \setminus \mathit{MEM}_I
\end{array}$$

[Lemma K.2]

$$
\begin{array}{l}
= P. \\
\quad \mathbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \land \mathit{inv}(b(v_0))\} \bullet \\
\quad\quad \left(\begin{array}{l}
\left(\begin{array}{l}
\mathit{get}.v_0?vv_0 \rightarrow \\
\mathbf{if}\ g_0(vv_0) \rightarrow\ \Omega_A(A_0) \\
[\!]\ \ldots \\
[\!]\ g_n(vv_0) \rightarrow\ \Omega_A(A_n) \\
\mathbf{fi}
\end{array}\right); \\
\mathit{terminate} \rightarrow \mathit{Skip} \\
[\![\emptyset \mid \mathit{MEM}_I \mid \{b\}]\!] \\
\left(\begin{array}{l}
(\Box\ \mathit{get}.n!b(n) \rightarrow \mathit{Memory}(b)) \\
\Box\ \left(\ \Box\ \mathit{set}.n?nv \rightarrow \mathit{Memory}(b \oplus \{n \mapsto nv\})\ \right) \\
\Box\ \mathit{terminate} \rightarrow \mathit{Skip}
\end{array}\right)
\end{array}\right) \\
\quad\quad \setminus \mathit{MEM}_I
\end{array}
$$

[Law 10]

provided

$\{\mathit{terminate}\} \subseteq \mathit{MEM}_I$

$\{\mathit{get}, \mathit{set}\} \subseteq \mathit{MEM}_I$

$\{\mathit{set}, \mathit{terminate}\} \in \mathit{MEM}_I$

$\{\mathit{set}, \mathit{terminate}\} \notin \{\mathit{get}\}$

$$
\begin{array}{l}
= P. \\
\quad \mathbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \land \mathit{inv}(b(v_0))\} \bullet \\
\quad\quad \left(\begin{array}{l}
\left(\begin{array}{l}
\mathit{get}.v_0?vv_0 \rightarrow \\
\mathbf{if}\ \rightarrow\ g_0(vv_0) \rightarrow\ \Omega_A(A_0) \\
[\!]\ \ldots \\
[\!]\ g_n(vv_0) \rightarrow\ \Omega_A(A_n) \\
\mathbf{fi}
\end{array}\right); \\
\mathit{terminate} \rightarrow \mathit{Skip} \\
[\![\emptyset \mid \mathit{MEM}_I \mid \{b\}]\!] \\
\left(\ \mathit{get}.vo!b(vo) \rightarrow \mathit{Memory}(b)\ \right)
\end{array}\right) \\
\quad\quad \setminus \mathit{MEM}_I
\end{array}
$$

[Law 24]

$\{\mathit{get}\} \subseteq \mathit{MEM}_I$

$v \notin \mathit{FV}(\mathit{get}.vo!b(vo) \rightarrow \mathit{Memory}(b))$

$= P.$

$$\begin{array}{l}\textbf{var } b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \wedge \mathit{inv}(b(v_0))\} \bullet \\ \quad \left(\begin{array}{l}\left(\begin{array}{l}\textbf{if } g_0(b(v_0)) \rightarrow \ \Omega_A(A_0) \\ \quad [\!] \ \ldots \\ \quad [\!] \ g_n(b(v_0)) \rightarrow \ \Omega_A(A_n) \\ \textbf{fi}\end{array}\right); \\ \mathit{terminate} \rightarrow \mathit{Skip} \\ [\![\emptyset \mid \mathit{MEM}_I \mid \{b\}]\!] \\ \mathit{Memory}(b)\end{array}\right) \\ \quad \setminus \mathit{MEM}_I\end{array}$$

From here, we have three possibilities:

- No alternative is true (Proved by Lemma K.8)
- Exactly one alternative is true (Proved by Lemma K.9)
- More than one alternative is true (Proved by Lemma K.10)

□

K.13 Assignment

Theorem K.13

$$\begin{array}{l}P_S.(x_0 := e_0(v_0)) \\ = \\ \Omega(P_S.(x_0 := e_0(v_0)))\end{array}$$

Proof.

$$\Omega(P_S.(x_0 := e_0(v_0)))$$

[Definition of Ω]

$$\begin{array}{l}= P. \\ \quad \textbf{var } b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \wedge \mathit{inv}(b(v_0))\} \bullet \\ \quad \left(\begin{array}{l}x_0 := e_0(v_0);\ \mathit{terminate} \rightarrow \mathit{Skip} \\ [\![\emptyset \mid \mathit{MEM}_I \mid \{b\}]\!] \\ \mathit{Memory}(b)\end{array}\right) \setminus \mathit{MEM}_I\end{array}$$

[Ω_A]

$$= P.$$
$$\quad \mathbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \land \mathit{inv}(b(v_0))\} \bullet$$
$$\quad\quad \left(\begin{array}{l} \left(\ get.v_0?vv_0 \to set.x_0!e_0(vv_0) \to \mathit{Skip}\ \right); \\ terminate \to \mathit{Skip} \\ \llbracket \emptyset \mid \mathit{MEM}_I \mid \{b\} \rrbracket \\ \mathit{Memory}(b) \end{array}\right) \setminus \mathit{MEM}_I$$

[Lemma K.2]

$$= P.$$
$$\quad \mathbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \land \mathit{inv}(b(v_0))\} \bullet$$
$$\quad\quad \left(\begin{array}{l} \left(\ get.v_0?vv_0 \to set.x_0!e_0(vv_0) \to \mathit{Skip}\ \right); \\ terminate \to \mathit{Skip} \\ \llbracket \emptyset \mid \mathit{MEM}_I \mid \{b\} \rrbracket \\ \left(\begin{array}{l} (\Box\ get.n!b(n) \to \mathit{Memory}(b)) \\ \Box\ \left(\ \Box\ set.n?nv \to \mathit{Memory}(b \oplus \{n \mapsto nv\})\ \right) \\ \Box\ terminate \to \mathit{Skip} \end{array}\right) \end{array}\right) \setminus \mathit{MEM}_I$$

[Law 10]

provided

$\{terminate\} \subseteq \mathit{MEM}_I$

$\{get, set\} \subseteq \mathit{MEM}_I$

$\{set, terminate\} \in \mathit{MEM}_I$

$\{set, terminate\} \notin \{get\}$

$$= P.$$
$$\quad \mathbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \land \mathit{inv}(b(v_0))\} \bullet$$
$$\quad\quad \left(\begin{array}{l} \left(\ get.v_0?vv_0 \to set.x_0!e_0(vv_0) \to \mathit{Skip}\ \right); \\ terminate \to \mathit{Skip} \\ \llbracket \emptyset \mid \mathit{MEM}_I \mid \{b\} \rrbracket \\ get.v_0!b(v_0) \to \mathit{Memory}(b) \end{array}\right)$$
$$\quad\quad \setminus \mathit{MEM}_I$$

[Law 24]

provided

$\{get\} \subseteq \mathit{MEM}_I$

$x \notin FV(\mathit{Memory}(b))$

$$= P.$$

$$\begin{array}{l}
\textbf{var}\ b : \{x : BINDING \mid b(v_0) \in T_0 \land inv(b(v_0))\} \bullet \\
\quad \left(\begin{array}{l}
\left(\ set.x_0!e_0(b(v_0)) \rightarrow Skip\ \right); \\
terminate \rightarrow Skip \\
\lbrack\!\lbrack \emptyset \mid MEM_I \mid \{b\} \rbrack\!\rbrack \\
Memory(b)
\end{array}\right) \\
\quad \setminus MEM_I
\end{array}$$

[Lemma K.2]

$= P.$

$$\begin{array}{l}
\textbf{var}\ b : \{x : BINDING \mid b(v_0) \in T_0 \land inv(b(v_0))\} \bullet \\
\quad \left(\begin{array}{l}
\textbf{var}\ b : BINDING \bullet \\
\qquad \left(\begin{array}{l}
\left(\ set.x_0!e_0(b(v_0)) \rightarrow Skip\ \right); \\
terminate \rightarrow Skip \\
\lbrack\!\lbrack \emptyset \mid MEM_I \mid \{b\} \rbrack\!\rbrack \\
\left(\begin{array}{l}
(\Box\ get.n!b(n) \rightarrow Memory(b)) \\
\Box \left(\ \Box\ set.n?nv \rightarrow Memory(b \oplus \{n \mapsto nv\})\ \right) \\
\Box\ terminate \rightarrow Skip
\end{array}\right)
\end{array}\right)
\end{array}\right) \\
\quad \setminus MEM_I
\end{array}$$

[Law 10]

provided

$\{terminate\} \subseteq MEM_I$

$\{get, set\} \subseteq MEM_I$

$\{set, terminate\} \in MEM_I$

$\{set, terminate\} \notin \{get\}$

$= P.$

$$\begin{array}{l}
\textbf{var}\ b : \{x : BINDING \mid b(v_0) \in T_0 \land inv(b(v_0))\} \bullet \\
\quad \left(\begin{array}{l}
\left(\ set.x_0!e_0(b(v_0)) \rightarrow Skip\ \right); \\
terminate \rightarrow Skip \\
\lbrack\!\lbrack \emptyset \mid MEM_I \mid \{b\} \rbrack\!\rbrack \\
(set.x_0?nv \rightarrow Memory(b \oplus \{x_0 \mapsto nv\}))
\end{array}\right) \\
\quad \setminus MEM_I
\end{array}$$

[Law 24]

provided

$\{set\} \subseteq MEM_I$

$x \notin FV(Memory(b))$

$= P.$

$$\begin{array}{l}
\textbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \land \mathit{inv}(b(v_0))\} \bullet \\
\quad \left(\begin{array}{l} \mathit{Skip};\ \mathit{terminate} \to \mathit{Skip} \\ [\![\emptyset \mid \mathit{MEM}_I \mid \{b\}]\!] \\ \mathit{Memory}(b \oplus \{x_0 \mapsto e_0(b(v_0))\}) \end{array}\right) \\
\quad \setminus \mathit{MEM}_I
\end{array}$$

[Law 8]

$$\begin{array}{l}
= P. \\
\quad \textbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \land \mathit{inv}(b(v_0))\} \bullet \\
\qquad \left(\begin{array}{l} \mathit{terminate} \to \mathit{Skip} \\ [\![\emptyset \mid \mathit{MEM}_I \mid \{b\}]\!] \\ \mathit{Memory}(b \oplus \{x_0 \mapsto e_0(b(v_0))\}) \end{array}\right) \\
\qquad \setminus \mathit{MEM}_I
\end{array}$$

[Lemma K.2]

$$\begin{array}{l}
= P. \\
\quad \textbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \land \mathit{inv}(b(v_0))\} \bullet \\
\qquad \left(\left(\begin{array}{l} \mathit{terminate} \to \mathit{Skip} \\ [\![\emptyset \mid \mathit{MEM}_I \mid \{b\}]\!] \\ \left(\begin{array}{l} (\Box\ \mathit{get}.n!\mathit{sb}(n) \to \mathit{Memory}(b)) \\ \Box\ \left(\ \Box\ \mathit{set}.n?\mathit{nv} \to \mathit{Memory}(b \oplus \{n \mapsto \mathit{nv}\})\ \right) \\ \Box\ \mathit{terminate} \to \mathit{Skip}; \end{array}\right) \end{array}\right)\right) \\
\qquad \setminus \mathit{MEM}_I
\end{array}$$

[Law 10]

provided

$\{\mathit{terminate}\} \subseteq \mathit{MEM}_I$

$\{\mathit{get}, \mathit{set}\} \subseteq \mathit{MEM}_I$

$\{\mathit{set}, \mathit{terminate}\} \in \mathit{MEM}_I$

$\{\mathit{set}, \mathit{terminate}\} \notin \{\mathit{get}\}$

$$\begin{array}{l}
= P. \\
\quad \textbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \land \mathit{inv}(b(v_0))\} \bullet \\
\qquad \left(\begin{array}{l} \mathit{terminate} \to \mathit{Skip} \\ [\![\emptyset \mid \mathit{MEM}_I \mid \{b\}]\!] \\ \mathit{terminate} \to \mathit{Skip} \end{array}\right) \\
\qquad \setminus \mathit{MEM}_I
\end{array}$$

[Law 25]

provided

$[\mathit{terminate} \in \mathit{MEM}_I]$

$= P.$

$$\textbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \wedge \mathit{inv}(b(v_0))\} \bullet$$
$$\left(\begin{array}{l} \mathit{Skip} \\ [\![\emptyset \mid \mathit{MEM}_I \mid \{b\}]\!] \\ \mathit{Skip} \end{array}\right)$$
$$\setminus\ \mathit{MEM}_I$$

[Law 28]

provided

$[\mathit{terminate} \in \mathit{MEM}_I]$

$= P.$

$$\textbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \wedge \mathit{inv}(b(v_0))\} \bullet$$
$$\mathit{Skip} \setminus \mathit{MEM}_I$$

[Law 15]

provided

$[\mathit{MEM}_I \cap \mathit{usedC}(\mathit{Skip}) = \emptyset]$

$= P.$

$$\textbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \wedge \mathit{inv}(b(v_0))\} \bullet$$
$$\mathit{Skip}$$

[Law 56]

$= P.$

$$\textbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \wedge \mathit{inv}(b(v_0))\} \bullet$$
$$b(x_0) := e_0(v_0)$$

[Law 26(b is the only component of S)]

$=$

$P_S.(b(x_0) := e_0(v_0))$

□

K.14 Sequential Composition

Theorem K.14

$$P_S.(A_1 ; A_2)$$
$$=$$
$$\Omega(P_S.(A_1 ; A_2))$$

Proof. Inductive Hypothesis: for any state S_1 and S_2

$$
\begin{array}{l}
(\mathbf{vres}\ x : \mathit{BINDING} \bullet A_1(x))(b) \\
= \\
\left(\begin{array}{l} (\Omega_A(A_2);\ \mathit{terminate} \to \mathit{Skip}) \\ [\![\emptyset \mid I\_MEM \mid \emptyset]\!] \\ \mathit{Memory}(b) \end{array}\right) \setminus \mathit{MEM}_I \\
\text{and} \\
(\mathbf{vres}\ x : \mathit{BINDING} \bullet A_2(x))(b) = \\
\left(\begin{array}{l} (\Omega_A(A_2);\ \mathit{terminate} \to \mathit{Skip}) \\ [\![\emptyset \mid I\_MEM \mid \emptyset]\!] \\ \mathit{Memory}(b) \end{array}\right) \setminus I\_MEM
\end{array}
$$

Proof.

$$
\begin{array}{ll}
\Omega(P_S.(A_1;\ A_2)) & [\Omega] \\
= P. & \\
\quad \mathbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \wedge \ldots\ \wedge \mathit{inv}(b(v_0), \ldots, b(v_n))\} \bullet & \\
\quad \left(\begin{array}{l} (\Omega_A(A_1;\ A_2);\ \mathit{terminate} \to \mathit{Skip}) \\ [\![\mathit{MEM}_I]\!] \\ \mathit{Memory}(b) \end{array}\right) \setminus \mathit{MEM}_I & \\
& [\Omega_A] \\
= P. & \\
\quad \mathbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \wedge \ldots\ \wedge \mathit{inv}(b(v_0), \ldots, b(v_n))\} \bullet & \\
\quad \left(\begin{array}{l} (\Omega_A(A_1);\ \Omega_A(A_2);\ \mathit{terminate} \to \mathit{Skip}) \\ [\![\mathit{MEM}_I]\!] \\ \mathit{Memory}(b) \end{array}\right) \setminus \mathit{MEM}_I & \\
& [\text{Lemma K.7}] \\
= P. \left(\begin{array}{l} \mathbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \wedge \ldots\ \wedge \mathit{inv}(b(v_0), \ldots, b(v_n))\} \bullet \\ \quad \left(\left(\begin{array}{l} (\Omega_A(A_1);\ \mathit{terminate} \to \mathit{Skip}) \\ [\![\emptyset \mid I\_MEM \mid \emptyset]\!] \\ \mathit{Memory}(b) \end{array}\right) \setminus I\_MEM \right); \\ \quad \left(\left(\begin{array}{l} (\Omega_A(A_2);\ \mathit{terminate} \to \mathit{Skip}) \\ [\![\emptyset \mid I\_MEM \mid \emptyset]\!] \\ \mathit{Memory}(b) \end{array}\right) \setminus I\_MEM \right) \end{array}\right) & \\
& [\text{IH}] \\
= P.\mathbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \wedge \ldots\ \wedge \mathit{inv}(b(v_0), \ldots, b(v_n))\} \bullet & \\
\quad \left(\begin{array}{l} (\mathbf{vres}\ x : \mathit{BINDING} \bullet A_1(x))(b); \\ (\mathbf{vres}\ x : \mathit{BINDING} \bullet A_2(x))(b) \end{array}\right) &
\end{array}
$$

[Semantics]

$$= P.\mathbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \land \ldots\ \land \mathit{inv}(b(v_0), \ldots, b(v_n))\} \bullet \left(\begin{array}{l}(\mathbf{var}\ x : \mathit{BINDING} \bullet x := b;\ A_1(x);\ b := x);\\ (\mathbf{var}\ x : \mathit{BINDING} \bullet x := b;\ A_2(x);\ b := x)\end{array}\right)$$

[Law 49]

$$= P.\mathbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \land \ldots\ \land \mathit{inv}(b(v_0), \ldots, b(v_n))\} \bullet \left(\begin{array}{l}(\mathbf{var}\ x : \mathit{BINDING} \bullet x := b;\ A_1(x);\ b := x);\\ (\mathbf{var}\ y : \mathit{BINDING} \bullet y := b;\ A_2(y);\ b := y)\end{array}\right)$$

[Law 8]

$$= P.\mathbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \land \ldots\ \land \mathit{inv}(b(v_0), \ldots, b(v_n))\} \bullet \left(\begin{array}{l}(\mathbf{var}\ x : \mathit{BINDING} \bullet x := b;\ A_1(x);\ b := x);\\ (\mathbf{var}\ y : \mathit{BINDING} \bullet y := b;\ A_2(y);\ b := y);\\ \mathit{Skip}\end{array}\right)$$

[Law 4]

$$= P.\mathbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \land \ldots\ \land \mathit{inv}(b(v_0), \ldots, b(v_n))\} \bullet \left(\begin{array}{l}\mathbf{var}\ y : \mathit{BINDING} \bullet\\ \qquad \left(\begin{array}{l}(\mathbf{var}\ x : \mathit{BINDING} \bullet x := b;\ A_1(x);\ b := x);\\ (y := b;\ A_2(y);\ b := y);\\ \mathit{Skip}\end{array}\right)\end{array}\right)$$

[Law 8]

$$= P.\mathbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \land \ldots\ \land \mathit{inv}(b(v_0), \ldots, b(v_n))\} \bullet \left(\begin{array}{l}\mathbf{var}\ y : \mathit{BINDING} \bullet\\ \qquad \left(\begin{array}{l}\mathit{Skip};\\ (\mathbf{var}\ x : \mathit{BINDING} \bullet x := b;\ A_1(x);\ b := x);\\ (y := b;\ A_2(y);\ b := y);\end{array}\right)\end{array}\right)$$

[Law 4]

$$= P.\mathbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \land \ldots\ \land \mathit{inv}(b(v_0), \ldots, b(v_n))\} \bullet \left(\begin{array}{l}\mathbf{var}\ y : \mathit{BINDING} \bullet\\ \qquad \left(\begin{array}{l}\mathbf{var}\ x : \mathit{BINDING} \bullet\\ \qquad \mathit{Skip};\\ \qquad (x := b;\ A_1(x);\ b := x);\\ \qquad (y := b;\ A_2(y);\ b := y);\end{array}\right)\end{array}\right)$$

[Law 8]

$$= P.\mathbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \wedge \ldots\ \wedge \mathit{inv}(b(v_0), \ldots, b(v_n))\} \bullet \left(\begin{array}{l} \mathbf{var}\ y : \mathit{BINDING} \bullet \\ \quad \left(\begin{array}{l} \mathbf{var}\ x : \mathit{BINDING} \bullet \\ \quad (x := b;\ A_1(x);\ b := x); \\ \quad (y := b;\ A_2(y);\ b := y); \end{array}\right) \end{array}\right)$$

[Law 7]

$$= P.\mathbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \wedge \ldots\ \wedge \mathit{inv}(b(v_0), \ldots, b(v_n))\} \bullet \left(\begin{array}{l} \mathbf{var}\ x, y : \mathit{BINDING} \bullet \\ \quad (x := b;\ A_1(x);\ b := x); \\ \quad (y := b;\ A_2(y);\ b := y); \end{array}\right)$$

[Law 47]

$$= P.\mathbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \wedge \ldots\ \wedge \mathit{inv}(b(v_0), \ldots, b(v_n))\} \bullet \left(\begin{array}{l} \mathbf{var}\ x, y : \mathit{BINDING} \bullet \\ \quad (x := b;\ A_1(x);\ b := x); \\ \quad (A_2(b);\ b := b); \end{array}\right)$$

[Laws 48 and 8]

$$= P.\mathbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \wedge \ldots\ \wedge \mathit{inv}(b(v_0), \ldots, b(v_n))\} \bullet \left(\begin{array}{l} \mathbf{var}\ x, y : \mathit{BINDING} \bullet \\ \quad (x := b;\ A_1(x);\ b := x); \\ \quad A_2(b) \end{array}\right)$$

[Law 47]

$$= P. \quad \mathbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \wedge \ldots\ \wedge \mathit{inv}(b(v_0), \ldots, b(v_n))\} \bullet \left(\begin{array}{l} \mathbf{var}\ x, y : \mathit{BINDING} \bullet \\ \quad (A_1(b);\ b := b); \\ \quad A_2(b) \end{array}\right)$$

[Laws 48 and 8]

$$= P.\mathbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \wedge \ldots\ \wedge \mathit{inv}(b(v_0), \ldots, b(v_n))\} \bullet \mathbf{var}\ x, y : \mathit{BINDING} \bullet A_1(b);\ A_2(b)$$

[Laws 7 and 6]

$$= P.\mathbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \wedge \ldots\ \wedge \mathit{inv}(b(v_0), \ldots, b(v_n))\} \bullet A_1(b);\ A_2(b)$$

[Law 26(b is the only component of S)]

$$= P_S.(A_1;\ A_2)$$

□

K.15 Auxiliary Lemmas

Lemma K.1

$$\begin{array}{l} A_1; \\ (\mathbf{vres}\ x : \mathit{BINDING} \bullet A_2(x))(b) \\ = \\ \mathbf{var}\ x : \mathit{BINDING} \bullet \\ \qquad \left(\begin{array}{l} A_1; \\ A_2(b) \end{array} \right) \end{array}$$

Proof.

$$A_1;$$
$$(\mathbf{vres}\ x : \mathit{BINDING} \bullet A_2(x))(b)$$

[Semantics]

$$=$$

$$A_1;$$
$$(\mathbf{var}\ x : \mathit{BINDING} \bullet x := b;\ A_2(x);\ b := x)$$

[Law 8]

$$=$$

$$A_1;$$
$$(\mathbf{var}\ x : \mathit{BINDING} \bullet x := b;\ A_2(x);\ b := x);$$
$$\mathit{Skip}$$

[Law 4]

$$=$$

$$\begin{array}{l} \mathbf{var}\ x : \mathit{BINDING} \bullet \\ \qquad \left(\begin{array}{l} A_1; \\ x := b;\ A_2(x);\ b := x; \\ \mathit{Skip} \end{array} \right) \end{array}$$

[Law 8]

$$=$$

$$\begin{array}{l} \mathbf{var}\ x : \mathit{BINDING} \bullet \\ \qquad \left(\begin{array}{l} A_1; \\ x := b;\ A_2(x);\ b := x \end{array} \right) \end{array}$$

[Law 47]

$$=$$

$$\mathbf{var}\ x : BINDING \bullet \left(\begin{array}{l} A_1; \\ A_2(b);\ b := b \end{array} \right)$$

[Laws 48 and 8]

$$=$$

$$\mathbf{var}\ x : BINDING \bullet \left(\begin{array}{l} A_1; \\ A_2(b) \end{array} \right)$$

Lemma K.2

$$\left(\begin{array}{l} A \\ [\![\emptyset \mid MEM_I \mid \{b\}]\!] \\ Memory(b) \end{array} \right) \setminus MEM_I$$
$$=$$
$$\left(\begin{array}{l} A \\ [\![\emptyset \mid MEM_I \mid \{b\}]\!] \\ \left(\begin{array}{l} (\Box\, n : NAME \bullet get.n!b(n) \rightarrow Cell(b)) \\ \Box\ (\Box\, n : NAME \bullet set.n?nv \rightarrow Cell(b \oplus \{n \mapsto nv\})) \\ \Box\ terminate \rightarrow Skip \end{array} \right) \end{array} \right) \setminus MEM_I$$

provided

- $b \notin FV(A)$

$$\left(\begin{array}{l} A \\ [\![\emptyset \mid MEM_I \mid \{b\}]\!] \\ Memory(b) \end{array} \right) \setminus MEM_I \qquad \text{[Law 49]}$$

$$= \left(\begin{array}{l} A \\ [\![\emptyset \mid MEM_I \mid \{bs\}]\!] \\ Memory(bs) \end{array} \right) \setminus MEM_I \qquad \text{[Law 9]}$$

$$= \left(\begin{array}{l} A \\ [\![\emptyset \mid MEM_I \mid \{bs\}]\!] \\ \left(\begin{array}{l} \mathbf{vres}\ b : BINDING \bullet \\ \qquad (\Box\, n : NAME \bullet get.n!b(n) \rightarrow Cell(b)) \\ \qquad \Box\, (\Box\, n : NAME \bullet set.n?nv \rightarrow Cell(b \oplus \{n \mapsto nv\})) \\ \qquad \Box\, terminate \rightarrow Skip \end{array} \right) (bs) \end{array} \right) \setminus MEM_I$$

[Semantics of **vres**]

$$= \left(\begin{array}{l} A \\ [\![\emptyset \mid MEM_I \mid \{bs\}]\!] \\ \left(\begin{array}{l} \mathbf{var}\ b : BINDING \bullet \\ \qquad b := bs; \\ \qquad \left(\begin{array}{l} (\Box\, n : NAME \bullet get.n!b(n) \rightarrow Cell(b)) \\ \Box\, (\Box\, n : NAME \bullet set.n?nv \rightarrow Cell(b \oplus \{n \mapsto nv\})) \\ \Box\, terminate \rightarrow Skip \end{array} \right); \\ \qquad bs := b \end{array} \right) \end{array} \right) \setminus MEM_I$$

[Law 1]

$$= \left(\begin{array}{l} \mathbf{var}\ b : BINDING \bullet \\ \qquad A \\ \qquad [\![\emptyset \mid MEM_I \mid \{bs\}]\!] \\ \qquad \left(\begin{array}{l} b := bs; \\ \left(\begin{array}{l} (\Box\, n : NAME \bullet get.n!b(n) \rightarrow Cell(b)) \\ \Box\, (\Box\, n : NAME \bullet set.n?nv \rightarrow Cell(b \oplus \{n \mapsto nv\})) \\ \Box\, terminate \rightarrow Skip \end{array} \right); \\ bs := b \end{array} \right) \end{array} \right) \setminus MEM_I$$

[Law 47]

$$= \left(\begin{array}{l} \mathbf{var}\ b : BINDING \bullet \\ \qquad A \\ \qquad [\![\emptyset \mid MEM_I \mid \{bs\}]\!] \\ \qquad \left(\begin{array}{l} \left(\begin{array}{l} (\Box\, n : NAME \bullet get.n!bs(n) \rightarrow Cell(bs)) \\ \Box\, (\Box\, n : NAME \bullet set.n?nv \rightarrow Cell(bs \oplus \{n \mapsto nv\})) \\ \Box\, terminate \rightarrow Skip \end{array} \right); \\ bs := bs \end{array} \right) \end{array} \right) \setminus MEM_I$$

[Laws 48 and 8]

$$= \left(\begin{array}{l} \mathbf{var}\ b : \mathit{BINDING} \bullet \\ \quad A \\ \quad [\![\emptyset \mid \mathit{MEM}_I \mid \{bs\}]\!] \\ \quad \left(\begin{array}{l} (\Box\, n : \mathit{NAME} \bullet \mathit{get}.n!bs(n) \to \mathit{Cell}(bs)) \\ \Box\ (\Box\, n : \mathit{NAME} \bullet \mathit{set}.n?nv \to \mathit{Cell}(bs \oplus \{n \mapsto nv\})) \\ \Box\ \mathit{terminate} \to \mathit{Skip} \end{array}\right) \end{array}\right) \setminus \mathit{MEM}_I$$

[Laws 6]

$$= \left(\begin{array}{l} A \\ [\![\emptyset \mid \mathit{MEM}_I \mid \{bs\}]\!] \\ \left(\begin{array}{l} (\Box\, n : \mathit{NAME} \bullet \mathit{get}.n!bs(n) \to \mathit{Cell}(bs)) \\ \Box\ (\Box\, n : \mathit{NAME} \bullet \mathit{set}.n?nv \to \mathit{Cell}(bs \oplus \{n \mapsto nv\})) \\ \Box\ \mathit{terminate} \to \mathit{Skip} \end{array}\right) \end{array}\right) \setminus \mathit{MEM}_I$$

[Law 49]

$$= \left(\begin{array}{l} A \\ [\![\emptyset \mid \mathit{MEM}_I \mid \{b\}]\!] \\ \left(\begin{array}{l} (\Box\, n : \mathit{NAME} \bullet \mathit{get}.n!b(n) \to \mathit{Cell}(b)) \\ \Box\ (\Box\, n : \mathit{NAME} \bullet \mathit{set}.n?nv \to \mathit{Cell}(b \oplus \{n \mapsto nv\})) \\ \Box\ \mathit{terminate} \to \mathit{Skip} \end{array}\right) \end{array}\right) \setminus \mathit{MEM}_I$$

Lemma K.3

$$\left(\begin{array}{l} \left(\left(\begin{array}{l} (A_1;\ \mathit{terminate} \to \mathit{Skip}) \\ [\![\emptyset \mid \mathit{MEM}_I \mid \{b\}]\!] \\ \mathit{Memory}(b) \end{array}\right) \setminus \mathit{MEM}_I\right); \\ \left(\left(\begin{array}{l} (A_2;\ \mathit{terminate} \to \mathit{Skip}) \\ [\![\emptyset \mid \mathit{MEM}_I \mid \{b\}]\!] \\ \mathit{Memory}(b) \end{array}\right) \setminus \mathit{MEM}_I\right) \end{array}\right)$$

$$= \left(\begin{array}{l} (A_1;\ A_2;\ \mathit{terminate} \to \mathit{Skip}) \\ [\![\mathit{MEM}_I]\!] \\ \mathit{Memory}(b) \end{array}\right) \setminus \mathit{MEM}_I$$

provided

- $\mathit{MEM}_I \cap \mathit{usedC}(A_1) = \emptyset$
- $b \notin \mathit{wrtV}(A_1)$

$$= \left(\begin{array}{l}\left(\left(\begin{array}{l}(A_1;\ \mathit{terminate} \to \mathit{Skip})\\ \lbrack\!\lbrack \emptyset \mid \mathit{MEM}_I \mid \{b\} \rbrack\!\rbrack\\ \mathit{Memory}(b)\end{array}\right) \setminus \mathit{MEM}_I\right);\\ \left(\left(\begin{array}{l}(A_2;\ \mathit{terminate} \to \mathit{Skip})\\ \lbrack\!\lbrack \emptyset \mid \mathit{MEM}_I \mid \{b\} \rbrack\!\rbrack\\ \mathit{Memory}(b)\end{array}\right) \setminus \mathit{MEM}_I\right)\end{array}\right)$$

[Law 30]

$[\mathit{initials}(\mathit{Memory}(b)) \subseteq \mathit{MEM}_I]$ *is true*

$[\mathit{MEM}_I \cap \mathit{usedC}(A_1) = \emptyset]$ *proviso*

$[\mathit{wrtV}(A_1) \cap \{b\} = \emptyset]$ *proviso*

$[\mathit{Memory}(b)$ is divergence-free$]$ *is true*

$[\{b\} \subseteq \{b\}]$ *is true*

$$= \left(\begin{array}{l}\left(\left(\begin{array}{l}A_1;\\ \left(\begin{array}{l}(\mathit{terminate} \to \mathit{Skip})\\ \lbrack\!\lbrack \emptyset \mid \mathit{MEM}_I \mid \{b\} \rbrack\!\rbrack\\ \mathit{Memory}(b)\end{array}\right)\end{array}\right) \setminus \mathit{MEM}_I\right);\\ \left(\left(\begin{array}{l}(A_2;\ \mathit{terminate} \to \mathit{Skip})\\ \lbrack\!\lbrack \emptyset \mid \mathit{MEM}_I \mid \{b\} \rbrack\!\rbrack\\ \mathit{Memory}(b)\end{array}\right) \setminus \mathit{MEM}_I\right)\end{array}\right)$$

[Laws 31 and 15 by proviso]

$$= \left(\begin{array}{l}\left(\begin{array}{l}A_1;\\ \left(\begin{array}{l}(\mathit{terminate} \to \mathit{Skip})\\ \lbrack\!\lbrack \emptyset \mid \mathit{MEM}_I \mid \{b\} \rbrack\!\rbrack\\ \mathit{Memory}(b)\end{array}\right) \setminus \mathit{MEM}_I\end{array}\right);\\ \left(\left(\begin{array}{l}(A_2;\ \mathit{terminate} \to \mathit{Skip})\\ \lbrack\!\lbrack \emptyset \mid \mathit{MEM}_I \mid \{b\} \rbrack\!\rbrack\\ \mathit{Memory}(b)\end{array}\right) \setminus \mathit{MEM}_I\right)\end{array}\right)$$

[Lemma K.2]

$$= \left(\begin{array}{l}A_1;\\ \left(\begin{array}{l}(\mathit{terminate} \to \mathit{Skip})\\ \lbrack\!\lbrack \emptyset \mid \mathit{MEM}_I \mid \{b\} \rbrack\!\rbrack\\ \left(\begin{array}{l}(\Box\, n : \mathit{NAME} \bullet\\ \qquad \mathit{get}.n!b(n) \to \mathit{Memory}(b))\\ \Box\ (\Box\, n : \mathit{NAME} \bullet\\ \qquad \mathit{set}.n?\mathit{nv} \to \mathit{Memory}(b \oplus \{n \mapsto \mathit{nv}\}))\\ \Box\ \mathit{terminate} \to \mathit{Skip}\end{array}\right)\end{array}\right)\\ \quad \setminus \mathit{MEM}_I\end{array}\right);$$

$$\left(\left(\begin{array}{l} (A_2;\ terminate \rightarrow Skip) \\ [\![\emptyset \mid MEM_I \mid \{b\}]\!] \\ Memory(b) \end{array} \right) \setminus MEM_I \right)$$

[Law 10]

provided

$\{terminate\} \subseteq MEM_I$

$\{get, set\} \subseteq MEM_I$

$\{get, set\} \cap \{terminate\} = \emptyset$

$$= \left(\begin{array}{l} A_1; \\ \left(\begin{array}{l} (terminate \rightarrow Skip) \\ [\![\emptyset \mid MEM_I \mid \{b\}]\!] \\ (terminate \rightarrow Skip) \end{array} \right) \\ \quad \setminus MEM_I \end{array} \right);$$

$$\left(\left(\begin{array}{l} (A_2;\ terminate \rightarrow Skip) \\ [\![\emptyset \mid MEM_I \mid \{b\}]\!] \\ Memory(b) \end{array} \right) \setminus MEM_I \right)$$

[Law 25]

provided

$[terminate \in MEM_I]$

$$= \left(\begin{array}{l} A_1; \\ (Skip\ [\![\emptyset \mid MEM_I \mid \{b\}]\!]\ Skip) \setminus MEM_I \end{array} \right);$$

$$\left(\left(\begin{array}{l} (A_2;\ terminate \rightarrow Skip) \\ [\![\emptyset \mid MEM_I \mid \{b\}]\!] \\ Memory(b) \end{array} \right) \setminus MEM_I \right)$$

[Law 28 and 15]

provided

$[terminate \in MEM_I]$

$$= (A_1;\ Skip);$$

$$\left(\left(\begin{array}{l} (A_2;\ terminate \rightarrow Skip) \\ [\![\emptyset \mid MEM_I \mid \{b\}]\!] \\ Memory(b) \end{array} \right) \setminus MEM_I \right)$$

[Law 8]

$$= \left(\begin{array}{l} A_1; \\ \left(\begin{array}{l} (A_2;\ terminate \rightarrow Skip) \\ [\![MEM_I]\!] \\ Memory(b) \end{array} \right) \end{array} \right) \setminus MEM_I$$

[Law 30 by proviso]

$$= \left(\begin{array}{l}(A_1;\ A_2;\ terminate \rightarrow Skip)\\ \lbrack\!\lbrack MEM_I \rbrack\!\rbrack\\ Memory(b)\end{array}\right) \setminus MEM_I$$

$[\Omega_A]$

$$= \left(\begin{array}{l}(A_1;\ A_2;\ terminate \rightarrow Skip)\\ \lbrack\!\lbrack MEM_I \rbrack\!\rbrack\\ Memory(b)\end{array}\right) \setminus MEM_I$$

Lemma K.4

$$\left(\begin{array}{l}\left(\left(\begin{array}{l}(A_1;\ A_2;\ terminate \rightarrow Skip)\\ \lbrack\!\lbrack \emptyset \mid MEM_I \mid \{b\} \rbrack\!\rbrack\\ Memory(b)\end{array}\right) \setminus MEM_I\right);\\ \left(\left(\begin{array}{l}(A_3;\ terminate \rightarrow Skip)\\ \lbrack\!\lbrack \emptyset \mid MEM_I \mid \{b\} \rbrack\!\rbrack\\ Memory(b)\end{array}\right) \setminus MEM_I\right)\end{array}\right)$$
$$=$$
$$\left(\begin{array}{l}(A_1;\ A_2;\ A_3;\ terminate \rightarrow Skip)\\ \lbrack\!\lbrack MEM_I \rbrack\!\rbrack\\ Memory(b)\end{array}\right) \setminus MEM_I$$

provided

- $MEM_I \cap usedC(A_1) = \emptyset$
- $b \notin wrtV(A_1)$
- $$\left(\begin{array}{l}\left(\left(\begin{array}{l}(A_2;\ terminate \rightarrow Skip)\\ \lbrack\!\lbrack \emptyset \mid MEM_I \mid \{b\} \rbrack\!\rbrack\\ Memory(b)\end{array}\right) \setminus MEM_I\right);\\ \left(\left(\begin{array}{l}(A_3;\ terminate \rightarrow Skip)\\ \lbrack\!\lbrack \emptyset \mid MEM_I \mid \{b\} \rbrack\!\rbrack\\ Memory(b)\end{array}\right) \setminus MEM_I\right)\end{array}\right)$$
  $$=$$
  $$\left(\begin{array}{l}(A_2;\ A_3;\ terminate \rightarrow Skip)\\ \lbrack\!\lbrack MEM_I \rbrack\!\rbrack\\ Memory(b)\end{array}\right) \setminus MEM_I$$

$$\left(\begin{array}{l}\left(\left(\begin{array}{l}(A_1;\ A_2;\ terminate \rightarrow Skip)\\ [\![\emptyset \mid MEM_I \mid \{b\}]\!]\\ Memory(b)\end{array}\right) \setminus MEM_I\right);\\ \left(\left(\begin{array}{l}(A_3;\ terminate \rightarrow Skip)\\ [\![\emptyset \mid MEM_I \mid \{b\}]\!]\\ Memory(b)\end{array}\right) \setminus MEM_I\right)\end{array}\right)$$

[Law 30]

$[initials(Memory) \subseteq MEM_I]$ *is true*

$[MEM_I \cap usedC(A_1) = \emptyset]$ *by proviso*

$[wrtV(A_1) \cap \{b\} = \emptyset]$ *by proviso*

$[Memory$ is divergence-free$]$ *is true*

$$= \begin{array}{l}\left(\left(\begin{array}{l}A_1;\\ \left(\begin{array}{l}(A_2;\ terminate \rightarrow Skip)\\ [\![\emptyset \mid MEM_I \mid \{b\}]\!]\\ Memory(b)\end{array}\right)\end{array}\right) \setminus MEM_I\right);\\ \left(\left(\begin{array}{l}(A_3;\ terminate \rightarrow Skip)\\ [\![\emptyset \mid MEM_I \mid \{b\}]\!]\\ Memory(b)\end{array}\right) \setminus MEM_I\right)\end{array}$$

[Laws 31 and 15]

$$\begin{array}{l}= A_1;\\ \quad\left(\left(\begin{array}{l}(A_2;\ terminate \rightarrow Skip)\\ [\![\emptyset \mid MEM_I \mid \{b\}]\!]\\ Memory(b)\end{array}\right) \setminus MEM_I\right);\\ \quad\left(\left(\begin{array}{l}(A_3;\ terminate \rightarrow Skip)\\ [\![\emptyset \mid MEM_I \mid \{b\}]\!]\\ Memory(b)\end{array}\right) \setminus MEM_I\right)\end{array}$$

[Proviso]

$$\begin{array}{l}= A_1;\\ \quad\left(\left(\begin{array}{l}(A_2;\ A_3;\ terminate \rightarrow Skip)\\ [\![MEM_I]\!]\\ Memory(b)\end{array}\right)\right) \setminus MEM_I\end{array}$$

[Laws 31 and 15]

$$= \left(\begin{array}{l}A_1;\\ \left(\begin{array}{l}(A_2;\ A_3;\ terminate \rightarrow Skip)\\ [\![MEM_I]\!]\\ Memory(b)\end{array}\right)\end{array}\right) \setminus MEM_I$$

[Law 30]

$$= \left(\begin{array}{l} (A_1;\ A_2;\ A_3;\ terminate \to Skip) \\ \llbracket MEM_I \rrbracket \\ Memory(b) \end{array}\right) \setminus MEM_I$$

Lemma K.5

$$\left(\begin{array}{l} \left(\left(\begin{array}{l} ((get.x?vv_0 \to A_1(vv_0));\ terminate \to Skip) \\ \llbracket \emptyset \mid MEM_I \mid \{b\} \rrbracket \\ Memory(b) \end{array}\right) \setminus MEM_I\right); \\ \left(\left(\begin{array}{l} (A_2;\ terminate \to Skip) \\ \llbracket \emptyset \mid MEM_I \mid \{b\} \rrbracket \\ Memory(b) \end{array}\right) \setminus MEM_I\right) \end{array}\right)$$
$$=$$
$$\left(\begin{array}{l} \left(\left(\begin{array}{l} (A_1(b(x));\ terminate \to Skip) \\ \llbracket \emptyset \mid MEM_I \mid \{b\} \rrbracket \\ Memory(b) \end{array}\right) \setminus MEM_I\right); \\ \left(\left(\begin{array}{l} (A_2;\ terminate \to Skip) \\ \llbracket \emptyset \mid MEM_I \mid \{b\} \rrbracket \\ Memory(b) \end{array}\right) \setminus MEM_I\right) \end{array}\right)$$

Proof.

$$\left(\begin{array}{l} \left(\left(\begin{array}{l} ((get.x?vv_0 \to A_1(vv_0));\ terminate \to Skip) \\ \llbracket \emptyset \mid MEM_I \mid \{b\} \rrbracket \\ Memory(b) \end{array}\right) \setminus MEM_I\right); \\ \left(\left(\begin{array}{l} A_2 \\ \llbracket \emptyset \mid MEM_I \mid \{b\} \rrbracket \\ Memory(b) \end{array}\right) \setminus MEM_I\right) \end{array}\right)$$

[Law 50]

$$= \left(\begin{array}{l} \left(\left(\begin{array}{l} (get.x?vv_0 \to (A_1(vv_0);\ terminate \to Skip)) \\ \llbracket \emptyset \mid MEM_I \mid \{b\} \rrbracket \\ Memory(b) \end{array}\right) \setminus MEM_I\right); \\ \left(\left(\begin{array}{l} A_2 \\ \llbracket \emptyset \mid MEM_I \mid \{b\} \rrbracket \\ Memory(b) \end{array}\right) \setminus MEM_I\right) \end{array}\right)$$

[Lemma K.2]

$$= \left(\begin{array}{l} \left(\left(\begin{array}{l} (get.x?vv_0 \to (A_1(vv_0);\ terminate \to Skip)) \\ [\![\emptyset \mid MEM_I \mid \{b\}]\!] \\ \left(\begin{array}{l} (\Box\, n : NAME \bullet get.n!b(n) \to Memory(b)) \\ \Box\ (\Box\, n : NAME \bullet set.n?nv \to Memory(b \oplus \{n \mapsto nv\})) \\ \Box\ terminate \to Skip \end{array}\right) \end{array}\right) \setminus MEM_I \right); \\ \left(\left(\begin{array}{l} A_2 \\ [\![\emptyset \mid MEM_I \mid \{b\}]\!] \\ Memory(b) \end{array}\right) \setminus MEM_I \right) \end{array}\right)$$

[Law 10]

$$= \left(\begin{array}{l} \left(\left(\begin{array}{l} (get.x?vv_0 \to (A_1(vv_0);\ terminate \to Skip)) \\ [\![\emptyset \mid MEM_I \mid \{b\}]\!] \\ (get.x!b(x) \to Memory(b)) \end{array}\right) \setminus MEM_I \right); \\ \left(\left(\begin{array}{l} A_2 \\ [\![\emptyset \mid MEM_I \mid \{b\}]\!] \\ Memory(b) \end{array}\right) \setminus MEM_I \right) \end{array}\right)$$

[Law 24]

$$= \left(\begin{array}{l} \left(\left(\begin{array}{l} (A_1(b(x));\ terminate \to Skip) \\ [\![\emptyset \mid MEM_I \mid \{b\}]\!] \\ Memory(b) \end{array}\right) \setminus MEM_I \right); \\ \left(\left(\begin{array}{l} A_2 \\ [\![\emptyset \mid MEM_I \mid \{b\}]\!] \\ Memory(b) \end{array}\right) \setminus MEM_I \right) \end{array}\right)$$

Lemma K.6

$$\left(\begin{array}{l} \left(\left(\begin{array}{l} ((set.x!e(b(x)) \to A);\ terminate \to Skip) \\ [\![\emptyset \mid MEM_I \mid \{b\}]\!] \\ Memory(b) \end{array}\right) \setminus MEM_I \right); \\ \left(\left(\begin{array}{l} (A_2;\ terminate \to Skip) \\ [\![\emptyset \mid MEM_I \mid \{b\}]\!] \\ Memory(b) \end{array}\right) \setminus MEM_I \right) \end{array}\right)$$

$$=$$

$$\left(\begin{array}{l} \left(\left(\begin{array}{l} (A;\ terminate \to Skip) \\ [\![\emptyset \mid MEM_I \mid \{b\}]\!] \\ Memory(b \oplus \{x \mapsto b(x)\}) \end{array}\right) \setminus MEM_I \right); \\ \left(\left(\begin{array}{l} (A_2;\ terminate \to Skip) \\ [\![\emptyset \mid MEM_I \mid \{b\}]\!] \\ Memory(b) \end{array}\right) \setminus MEM_I \right) \end{array}\right)$$

Proof.

$$\left(\begin{array}{l}\left(\left(\begin{array}{l}((set.x!e(b(x)) \rightarrow A);\ terminate \rightarrow Skip)\\ \llbracket \emptyset \mid MEM_I \mid \{b\} \rrbracket\\ Memory(b)\end{array}\right) \setminus MEM_I\right);\\ \left(\left(\begin{array}{l}A_2\\ \llbracket \emptyset \mid MEM_I \mid \{b\} \rrbracket\\ Memory(b)\end{array}\right) \setminus MEM_I\right)\end{array}\right)$$

[Law 34, 50 and Associtivity]

$$= \left(\begin{array}{l}\left(\left(\begin{array}{l}(set.x!e(b(x)) \rightarrow (A;\ terminate \rightarrow Skip))\\ \llbracket \emptyset \mid MEM_I \mid \{b\} \rrbracket\\ Memory(b)\end{array}\right) \setminus MEM_I\right);\\ \left(\left(\begin{array}{l}A_2\\ \llbracket \emptyset \mid MEM_I \mid \{b\} \rrbracket\\ Memory(b)\end{array}\right) \setminus MEM_I\right)\end{array}\right)$$

[Lemma K.2]

$$= \left(\begin{array}{l}\left(\left(\begin{array}{l}(set.x!e(b(x)) \rightarrow (A;\ terminate \rightarrow Skip))\\ \llbracket \emptyset \mid MEM_I \mid \{b\} \rrbracket\\ \left(\begin{array}{l}(\Box\, n : NAME \bullet get.n!b(n) \rightarrow Memory(b))\\ \Box\ (\Box\, n : NAME \bullet set.n?nv \rightarrow Memory(b \oplus \{n \mapsto nv\}))\\ \Box\ terminate \rightarrow Skip\end{array}\right)\end{array}\right) \setminus MEM_I\right);\\ \left(\left(\begin{array}{l}A_2\\ \llbracket \emptyset \mid MEM_I \mid \{b\} \rrbracket\\ Memory(b)\end{array}\right) \setminus MEM_I\right)\end{array}\right)$$

[Law 10]

$$= \left(\begin{array}{l}\left(\left(\begin{array}{l}(set.x!e(b(x)) \rightarrow (A;\ terminate \rightarrow Skip))\\ \llbracket \emptyset \mid MEM_I \mid \{b\} \rrbracket\\ (set.x?nv \rightarrow Memory(b \oplus \{x \mapsto nv\}))\end{array}\right) \setminus MEM_I\right);\\ \left(\left(\begin{array}{l}A_2\\ \llbracket \emptyset \mid MEM_I \mid \{b\} \rrbracket\\ Memory(b)\end{array}\right) \setminus MEM_I\right)\end{array}\right)$$

[Law 35]

$$= \left(\begin{array}{l}\left(\left(\begin{array}{l}set.x!e(b(x)) \rightarrow\\ \qquad \left(\begin{array}{l}(A;\ terminate \rightarrow Skip)\\ \llbracket \emptyset \mid MEM_I \mid \{b\} \rrbracket\\ Memory(b \oplus \{x \mapsto b(x)\})\end{array}\right)\end{array}\right) \setminus MEM_I\right);\\ \left(\left(\begin{array}{l}A_2\\ \llbracket \emptyset \mid MEM_I \mid \{b\} \rrbracket\\ Memory(b)\end{array}\right) \setminus MEM_I\right)\end{array}\right)$$

[Laws 34, 31 and 15]

$$= \left(\begin{array}{l} \left(\left(\begin{array}{l} (A;\ \mathit{terminate} \to \mathit{Skip}) \\ \lbrack\!\lbrack \emptyset \mid \mathit{MEM}_I \mid \{b\} \rbrack\!\rbrack \\ \mathit{Memory}(b \oplus \{x \mapsto b(x)\}) \end{array} \right) \setminus \mathit{MEM}_I \right); \\ \left(\left(\begin{array}{l} (A_2;\ \mathit{terminate} \to \mathit{Skip}) \\ \lbrack\!\lbrack \emptyset \mid \mathit{MEM}_I \mid \{b\} \rbrack\!\rbrack \\ \mathit{Memory}(b) \end{array} \right) \setminus \mathit{MEM}_I \right) \end{array} \right)$$

Lemma K.7

$$\begin{array}{l} \left(\left(\begin{array}{l} (\Omega_A(A_1);\ \mathit{terminate} \to \mathit{Skip}) \\ \lbrack\!\lbrack \emptyset \mid \mathit{MEM}_I \mid \{b\} \rbrack\!\rbrack \\ \mathit{Memory}(b) \end{array} \right) \setminus \mathit{MEM}_I \right); \\ \left(\left(\begin{array}{l} (\Omega_A(A_2);\ \mathit{terminate} \to \mathit{Skip}) \\ \lbrack\!\lbrack \emptyset \mid \mathit{MEM}_I \mid \{b\} \rbrack\!\rbrack \\ \mathit{Memory}(b) \end{array} \right) \setminus \mathit{MEM}_I \right) \\ = \left(\begin{array}{l} (\Omega_A(A_1);\ \Omega_A(A_2);\ \mathit{terminate} \to \mathit{Skip}) \\ \lbrack\!\lbrack \mathit{MEM}_I \rbrack\!\rbrack \\ \mathit{Memory}(b) \end{array} \right) \setminus \mathit{MEM}_I \end{array}$$

Proof. The overall proof is by induction on the syntax of accepted actions for A_1. We consider *Skip*, *Stop*, *Chaos*, prefixing, guarded actions, and assignment.

Base cases:$\Omega_A(A_1)$ is one of the following actions

- *Skip*
- *Stop*
- *Chaos*
- $c \to \mathit{Skip}$ ($c \notin \mathit{MEM}_I$)

All these cases can be proved using the structure below.

$$\left(\begin{array}{l} \left(\left(\begin{array}{l} (\Omega_A(A_1);\ \mathit{terminate} \to \mathit{Skip}) \\ \lbrack\!\lbrack \emptyset \mid \mathit{MEM}_I \mid \{b\} \rbrack\!\rbrack \\ \mathit{Memory}(b) \end{array} \right) \setminus \mathit{MEM}_I \right); \\ \left(\left(\begin{array}{l} (\Omega_A(A_2);\ \mathit{terminate} \to \mathit{Skip}) \\ \lbrack\!\lbrack \emptyset \mid \mathit{MEM}_I \mid \{b\} \rbrack\!\rbrack \\ \mathit{Memory}(b) \end{array} \right) \setminus \mathit{MEM}_I \right) \end{array} \right)$$

[Ω_A]

$$= \left(\begin{array}{l} \left(\left(\begin{array}{l} (A_1;\ \mathit{terminate} \rightarrow \mathit{Skip}) \\ [\![\emptyset \mid \mathit{MEM}_I \mid \{b\}]\!] \\ \mathit{Memory}(b) \end{array} \right) \setminus \mathit{MEM}_I \right); \\ \left(\left(\begin{array}{l} (\Omega_A(A_2);\ \mathit{terminate} \rightarrow \mathit{Skip}) \\ [\![\emptyset \mid \mathit{MEM}_I \mid \{b\}]\!] \\ \mathit{Memory}(b) \end{array} \right) \setminus \mathit{MEM}_I \right) \end{array} \right)$$

[Lemma K.3]

$[\mathit{MEM}_I \cap \mathit{usedC}(A_1) = \emptyset]$

$[\mathit{wrtV}(A_1) \cap \{b\} = \emptyset]$

$$= \left(\begin{array}{l} (A_1;\ \Omega_A(A_2);\ \mathit{terminate} \rightarrow \mathit{Skip}) \\ [\![\mathit{MEM}_I]\!] \\ \mathit{Memory}(b) \end{array} \right) \setminus \mathit{MEM}_I$$

$[\Omega_A]$

$$= \left(\begin{array}{l} (\Omega_A(A_1);\ \Omega_A(A_2);\ \mathit{terminate} \rightarrow \mathit{Skip}) \\ [\![\mathit{MEM}_I]\!] \\ \mathit{Memory}(b) \end{array} \right) \setminus \mathit{MEM}_I$$

Inductive cases:

Proved:

K.15.1 $c \rightarrow A_1$

K.15.2 $c.e(v_0, \ldots, v_n, l_0, \ldots, l_m) \rightarrow A_1$

K.15.3 $c!e(v_0, \ldots, v_n, l_0, \ldots, l_m) \rightarrow A_1$

K.15.4 $g(v_0, \ldots, v_n, l_0, \ldots, l_m)\ \&\ A_1$

K.15.5 $c?x : P(x, v_0, \ldots, v_n, l_0, \ldots, l_m) \rightarrow A_1$

K.15.6 $x_0, \ldots, x_n := e_0(v_0, \ldots, v_n, l_0, \ldots, l_m), \ldots, e_n(v_0, \ldots, v_n, l_0, \ldots, l_m)$

To prove:

1. $A_1;\ A_2$ (IHs)
2. $A_1 \sqcap A_2$ (IHs)
3. $A_1 \square A_2$ (*gets* and IHs)
4. $A_1 [\![ns_1 \mid cs \mid ns_2]\!] A_2$ (*gets*, IHs, and much more)
5. $A_1 |\!|\![ns_1 \mid ns_2]\!|\!| A_2$ (free lunch)
6. $A \setminus cs$ (IHs)

7. $(x : T \bullet A(x))(e)$ (IHs)
8. $\mu\, X \bullet A(X)$ (IHs)
9. $w : [\mathit{pre}(v_0, \ldots, v_n, l_0, \ldots, l_m), \mathit{post}(v_0, \ldots, v_n, l_0, \ldots, l_m)]$ (IHs)
10. $\{g\}$ (free lunch)
11. $[g]$ (free lunch)
12. $[\mathit{udecl};\ \mathit{ddecl}' \mid \mathit{pred}]$ (free lunch)
13. $A[\mathit{old}_1, \ldots, \mathit{old}_n := \mathit{new}_1, \ldots, \mathit{new}_n]$ (free lunch)
14. Iterated operators (induction on type using IH, but a free lunch)
15. **if** **_fi** (induction on number of guards using IH, but a free lunch)

Inductive hypothesis:

$$
\begin{array}{l}
\left(\begin{array}{l}(\Omega_A(A_1);\ \mathit{terminate} \rightarrow \mathit{Skip})\\ \lbrack\!\lbrack \emptyset \mid \mathit{MEM}_I \mid \{b\} \rbrack\!\rbrack\\ \mathit{Memory}(b)\end{array}\right) \setminus \mathit{MEM}_I;\\
\left(\begin{array}{l}(\Omega_A(A_2);\ \mathit{terminate} \rightarrow \mathit{Skip})\\ \lbrack\!\lbrack \emptyset \mid \mathit{MEM}_I \mid \{b\} \rbrack\!\rbrack\\ \mathit{Memory}(b)\end{array}\right) \setminus \mathit{MEM}_I\\
= \left(\begin{array}{l}(\Omega_A(A_1);\ \Omega_A(A_2);\ \mathit{terminate} \rightarrow \mathit{Skip})\\ \lbrack\!\lbrack \mathit{MEM}_I \rbrack\!\rbrack\\ \mathit{Memory}(b)\end{array}\right) \setminus \mathit{MEM}_I
\end{array}
$$

K.15.1 Free Event

For $c \notin \{\mathit{set}, \mathit{get}, \mathit{terminate}\}$:

$$
\begin{array}{l}
\left(\left(\begin{array}{l}(\Omega_A(c \rightarrow A_1);\ \mathit{terminate} \rightarrow \mathit{Skip})\\ \lbrack\!\lbrack \emptyset \mid \mathit{MEM}_I \mid \{b\} \rbrack\!\rbrack\\ \mathit{Memory}(b)\end{array}\right) \setminus \mathit{MEM}_I\right);\\
\left(\left(\begin{array}{l}(\Omega_A(A_2);\ \mathit{terminate} \rightarrow \mathit{Skip})\\ \lbrack\!\lbrack \emptyset \mid \mathit{MEM}_I \mid \{b\} \rbrack\!\rbrack\\ \mathit{Memory}(b)\end{array}\right) \setminus \mathit{MEM}_I\right)\\
= \left(\begin{array}{l}(\Omega_A(c \rightarrow A_1);\ \Omega_A(A_2);\ \mathit{terminate} \rightarrow \mathit{Skip})\\ \lbrack\!\lbrack \mathit{MEM}_I \rbrack\!\rbrack\\ \mathit{Memory}(b)\end{array}\right) \setminus \mathit{MEM}_I
\end{array}
$$

Proof.

$$\begin{pmatrix}\begin{pmatrix}(\Omega_A(c \to A_1);\ \mathit{terminate} \to \mathit{Skip})\\ \llbracket \emptyset \mid \mathit{MEM_I} \mid \{b\} \rrbracket\\ \mathit{Memory}(b)\end{pmatrix} \setminus \mathit{MEM_I}\end{pmatrix};$$
$$\begin{pmatrix}\begin{pmatrix}(\Omega_A(A_2);\ \mathit{terminate} \to \mathit{Skip})\\ \llbracket \emptyset \mid \mathit{MEM_I} \mid \{b\} \rrbracket\\ \mathit{Memory}(b)\end{pmatrix} \setminus \mathit{MEM_I}\end{pmatrix}$$

$[\Omega_A]$

$$= \begin{pmatrix}\begin{pmatrix}(c \to \Omega_A(A_1);\ \mathit{terminate} \to \mathit{Skip})\\ \llbracket \emptyset \mid \mathit{MEM_I} \mid \{b\} \rrbracket\\ \mathit{Memory}(b)\end{pmatrix} \setminus \mathit{MEM_I}\end{pmatrix};$$
$$\begin{pmatrix}\begin{pmatrix}(\Omega_A(A_2);\ \mathit{terminate} \to \mathit{Skip})\\ \llbracket \emptyset \mid \mathit{MEM_I} \mid \{b\} \rrbracket\\ \mathit{Memory}(b)\end{pmatrix} \setminus \mathit{MEM_I}\end{pmatrix}$$

[Semantics of $c \to A$]

$$= \begin{pmatrix}\begin{pmatrix}((c \to \mathit{Skip});\ \Omega_A(A_1);\ \mathit{terminate} \to \mathit{Skip})\\ \llbracket \emptyset \mid \mathit{MEM_I} \mid \{b\} \rrbracket\\ \mathit{Memory}(b)\end{pmatrix} \setminus \mathit{MEM_I}\end{pmatrix};$$
$$\begin{pmatrix}\begin{pmatrix}(\Omega_A(A_2);\ \mathit{terminate} \to \mathit{Skip})\\ \llbracket \emptyset \mid \mathit{MEM_I} \mid \{b\} \rrbracket\\ \mathit{Memory}(b)\end{pmatrix} \setminus \mathit{MEM_I}\end{pmatrix}$$

[Lemma K.4]

$[\mathit{MEM_I} \cap (c \to \mathit{Skip}) = \emptyset]$

$[\mathit{wrtV}(c \to A) \cap \{b\} = \emptyset]$

IH

$$= \begin{pmatrix}((c \to \mathit{Skip});\ \Omega_A(A_1);\ \Omega_A(A_2);\ \mathit{terminate} \to \mathit{Skip})\\ \llbracket \mathit{MEM_I} \rrbracket\\ \mathit{Memory}(b)\end{pmatrix} \setminus \mathit{MEM_I}$$

[Semantics of $c \to A$]

$$= \begin{pmatrix}((c \to \Omega_A(A_1));\ \Omega_A(A_2);\ \mathit{terminate} \to \mathit{Skip})\\ \llbracket \mathit{MEM_I} \rrbracket\\ \mathit{Memory}(b)\end{pmatrix} \setminus \mathit{MEM_I}$$

$[\Omega_A]$

$$= \begin{pmatrix}(\Omega_A(c \to A_1);\ \Omega_A(A_2);\ \mathit{terminate} \to \mathit{Skip})\\ \llbracket \mathit{MEM_I} \rrbracket\\ \mathit{Memory}(b)\end{pmatrix} \setminus \mathit{MEM_I}$$

K.15.2 Simple Synchronisation Event

$$\left(\left(\begin{array}{l}(\Omega_A(c.e(v_0,\ldots,v_n,l_0,\ldots,l_m)\rightarrow A_1);\ \mathit{terminate}\rightarrow \mathit{Skip})\\ \lbrack\!\lbrack \emptyset \mid \mathit{MEM}_I \mid \{b\}\rbrack\!\rbrack\\ \mathit{Memory}(b)\end{array}\right)\setminus \mathit{MEM}_I\right);$$
$$\left(\left(\begin{array}{l}(\Omega_A(A_2);\ \mathit{terminate}\rightarrow \mathit{Skip})\\ \lbrack\!\lbrack \emptyset \mid \mathit{MEM}_I \mid \{b\}\rbrack\!\rbrack\\ \mathit{Memory}(b)\end{array}\right)\setminus \mathit{MEM}_I\right)$$
$$=\left(\begin{array}{l}(\Omega_A(c.e(v_0,\ldots,v_n,l_0,\ldots,l_m)\rightarrow A_1);\ \Omega_A(A_2);\ \mathit{terminate}\rightarrow \mathit{Skip})\\ \lbrack\!\lbrack \mathit{MEM}_I\rbrack\!\rbrack\\ \mathit{Memory}(b)\end{array}\right)\setminus \mathit{MEM}_I$$

Proof. In this proof and those that follow we will consider a single state component x. The generalisation of this proof by induction on the number of state components is rather simple, but omitted here for the sake of presentation.

$$\left(\left(\begin{array}{l}(\Omega_A(c.e(x)\rightarrow A_1);\ \mathit{terminate}\rightarrow \mathit{Skip})\\ \lbrack\!\lbrack \emptyset \mid \mathit{MEM}_I \mid \{b\}\rbrack\!\rbrack\\ \mathit{Memory}(b)\end{array}\right)\setminus \mathit{MEM}_I\right);$$
$$\left(\left(\begin{array}{l}(\Omega_A(A_2);\ \mathit{terminate}\rightarrow \mathit{Skip})\\ \lbrack\!\lbrack \emptyset \mid \mathit{MEM}_I \mid \{b\}\rbrack\!\rbrack\\ \mathit{Memory}(b)\end{array}\right)\setminus \mathit{MEM}_I\right)$$

[Ω_A]

$$=\left(\begin{array}{l}\left(\left(\begin{array}{l}(\mathit{get}.x?vv_0\rightarrow c.e(vv_0)\rightarrow \Omega_A(A_1));\ \mathit{terminate}\rightarrow \mathit{Skip})\\ \lbrack\!\lbrack \emptyset \mid \mathit{MEM}_I \mid \{b\}\rbrack\!\rbrack\\ \mathit{Memory}(b)\end{array}\right)\setminus \mathit{MEM}_I\right);\\ \left(\left(\begin{array}{l}(\Omega_A(A_2);\ \mathit{terminate}\rightarrow \mathit{Skip})\\ \lbrack\!\lbrack \emptyset \mid \mathit{MEM}_I \mid \{b\}\rbrack\!\rbrack\\ \mathit{Memory}(b)\end{array}\right)\setminus \mathit{MEM}_I\right)\end{array}\right.$$

[Lemma K.5]

$$=\left(\begin{array}{l}\left(\left(\begin{array}{l}(c.e(b(x))\rightarrow \Omega_A(A_1));\ \mathit{terminate}\rightarrow \mathit{Skip})\\ \lbrack\!\lbrack \emptyset \mid \mathit{MEM}_I \mid \{b\}\rbrack\!\rbrack\\ \mathit{Memory}(x)\end{array}\right)\setminus \mathit{MEM}_I\right);\\ \left(\left(\begin{array}{l}(\Omega_A(A_2);\ \mathit{terminate}\rightarrow \mathit{Skip})\\ \lbrack\!\lbrack \emptyset \mid \mathit{MEM}_I \mid \{b\}\rbrack\!\rbrack\\ \mathit{Memory}(b)\end{array}\right)\setminus \mathit{MEM}_I\right)\end{array}\right.$$

[Case K.15.1]

$$= \left(\begin{array}{l} (\Omega_A(c.e(b(x)) \rightarrow \Omega_A(A_1));\ \Omega_A(A_2);\ \mathit{terminate} \rightarrow \mathit{Skip}) \\ [\![\mathit{MEM}_I]\!] \\ \mathit{Memory}(b) \end{array}\right) \setminus \mathit{MEM}_I$$

K.15.3 Output Event

$$\left(\left(\begin{array}{l} (\Omega_A(c!e(v_0,\ldots,v_n,l_0,\ldots,l_m) \rightarrow A_1);\ \mathit{terminate} \rightarrow \mathit{Skip}) \\ [\![\emptyset \mid \mathit{MEM}_I \mid \{b\}]\!] \\ \mathit{Memory}(b) \end{array}\right) \setminus \mathit{MEM}_I\right);$$
$$\left(\left(\begin{array}{l} (\Omega_A(A_2);\ \mathit{terminate} \rightarrow \mathit{Skip}) \\ [\![\emptyset \mid \mathit{MEM}_I \mid \{b\}]\!] \\ \mathit{Memory}(b) \end{array}\right) \setminus \mathit{MEM}_I\right)$$
$$= \left(\begin{array}{l} (\Omega_A(c!e(v_0,\ldots,v_n,l_0,\ldots,l_m) \rightarrow A_1);\ \Omega_A(A_2);\ \mathit{terminate} \rightarrow \mathit{Skip}) \\ [\![\mathit{MEM}_I]\!] \\ \mathit{Memory}(b) \end{array}\right) \setminus \mathit{MEM}_I$$

Proof.

$$\left(\left(\begin{array}{l} (\Omega_A(c!e(x) \rightarrow A_1);\ \mathit{terminate} \rightarrow \mathit{Skip}) \\ [\![\emptyset \mid \mathit{MEM}_I \mid \{b\}]\!] \\ \mathit{Memory}(b) \end{array}\right) \setminus \mathit{MEM}_I\right);$$
$$\left(\left(\begin{array}{l} (\Omega_A(A_2);\ \mathit{terminate} \rightarrow \mathit{Skip}) \\ [\![\emptyset \mid \mathit{MEM}_I \mid \{b\}]\!] \\ \mathit{Memory}(b) \end{array}\right) \setminus \mathit{MEM}_I\right)$$

$[c!e \rightarrow A]$

$$= \left(\left(\begin{array}{l} (\Omega_A(c.e(x) \rightarrow A_1);\ \mathit{terminate} \rightarrow \mathit{Skip}) \\ [\![\emptyset \mid \mathit{MEM}_I \mid \{b\}]\!] \\ \mathit{Memory}(b) \end{array}\right) \setminus \mathit{MEM}_I\right);$$
$$\left(\left(\begin{array}{l} (\Omega_A(A_2);\ \mathit{terminate} \rightarrow \mathit{Skip}) \\ [\![\emptyset \mid \mathit{MEM}_I \mid \{b\}]\!] \\ \mathit{Memory}(b) \end{array}\right) \setminus \mathit{MEM}_I\right)$$

[Case K.15.2]

$$= \left(\begin{array}{l} (\Omega_A(c.e(b(x)) \rightarrow A_1);\ \Omega_A(A_2);\ \mathit{terminate} \rightarrow \mathit{Skip}) \\ [\![\mathit{MEM}_I]\!] \\ \mathit{Memory}(b) \end{array}\right) \setminus \mathit{MEM}_I$$

$[c.e \rightarrow A]$

$$= \left(\begin{array}{l} (\Omega_A(c!e(b(x)) \rightarrow A_1);\ \Omega_A(A_2);\ \mathit{terminate} \rightarrow \mathit{Skip}) \\ [\![\mathit{MEM}_I]\!] \\ \mathit{Memory}(b) \end{array}\right) \setminus \mathit{MEM}_I$$

K.15.4 Guarded Action

$$\begin{array}{l}\left(\left(\begin{array}{l}(\Omega_A(g(v_0,\ldots,v_n,l_0,\ldots,l_m)\ \&\ A_1);\ terminate \rightarrow Skip)\\ \llbracket \emptyset \mid MEM_I \mid \{b\} \rrbracket\\ Memory(b)\end{array}\right) \setminus MEM_I\right);\\ \left(\left(\begin{array}{l}(\Omega_A(A_2);\ terminate \rightarrow Skip)\\ \llbracket \emptyset \mid MEM_I \mid \{b\} \rrbracket\\ Memory(b)\end{array}\right) \setminus MEM_I\right)\\ = \left(\begin{array}{l}(\Omega_A(g(v_0,\ldots,v_n,l_0,\ldots,l_m)\ \&\ A_1);\ \Omega_A(A_2);\ terminate \rightarrow Skip)\\ \llbracket MEM_I \rrbracket\\ Memory(b)\end{array}\right) \setminus MEM_I\end{array}$$

Proof.

$$\begin{array}{l}\left(\left(\begin{array}{l}(\Omega_A(g(v_0)\ \&\ A_1);\ terminate \rightarrow Skip)\\ \llbracket \emptyset \mid MEM_I \mid \{b\} \rrbracket\\ Memory(b)\end{array}\right) \setminus MEM_I\right);\\ \left(\left(\begin{array}{l}(\Omega_A(A_2);\ terminate \rightarrow Skip)\\ \llbracket \emptyset \mid MEM_I \mid \{b\} \rrbracket\\ Memory(b)\end{array}\right) \setminus MEM_I\right)\end{array}$$

[Ω_A]

$$\begin{array}{l}= \left(\left(\begin{array}{l}((get.x?vv_0 \rightarrow g(vv_0)\ \&\ \Omega_A(A_1));\ terminate \rightarrow Skip)\\ \llbracket \emptyset \mid MEM_I \mid \{b\} \rrbracket\\ Memory(b)\end{array}\right) \setminus MEM_I\right);\\ \left(\left(\begin{array}{l}(\Omega_A(A_2);\ terminate \rightarrow Skip)\\ \llbracket \emptyset \mid MEM_I \mid \{b\} \rrbracket\\ Memory(b)\end{array}\right) \setminus MEM_I\right)\end{array}$$

[Lemma K.5]

$$\begin{array}{l}= \left(\left(\begin{array}{l}((g(b(x))\ \&\ \Omega_A(A_1));\ terminate \rightarrow Skip)\\ \llbracket \emptyset \mid MEM_I \mid \{b\} \rrbracket\\ Memory(b)\end{array}\right) \setminus MEM_I\right);\\ \left(\left(\begin{array}{l}(\Omega_A(A_2);\ terminate \rightarrow Skip)\\ \llbracket \emptyset \mid MEM_I \mid \{b\} \rrbracket\\ Memory(b)\end{array}\right) \setminus MEM_I\right)\end{array}$$

[Laws 8 and 32]

$$= \left(\left(\begin{array}{l}((g(b(x)) \mathbin{\&} \mathit{Skip});\ \Omega_A(A_1);\ \mathit{terminate} \to \mathit{Skip})\\ \llbracket \emptyset \mid \mathit{MEM}_I \mid \{b\} \rrbracket\\ \mathit{Memory}(b)\end{array}\right) \setminus \mathit{MEM}_I\right);$$
$$\left(\left(\begin{array}{l}(\Omega_A(A_2);\ \mathit{terminate} \to \mathit{Skip})\\ \llbracket \emptyset \mid \mathit{MEM}_I \mid \{b\} \rrbracket\\ \mathit{Memory}(b)\end{array}\right) \setminus \mathit{MEM}_I\right)$$

[Lemma K.4]

provided

$[\mathit{MEM}_I \cap \emptyset = \emptyset]$

$[b \notin \emptyset]$

IH

$$= \left(\begin{array}{l}((g(b(x)) \mathbin{\&} \mathit{Skip});\ \Omega_A(A_1);\ \Omega_A(A_2);\ \mathit{terminate} \to \mathit{Skip})\\ \llbracket \emptyset \mid \mathit{MEM}_I \mid \{b\} \rrbracket\\ \mathit{Memory}(b)\end{array}\right) \setminus \mathit{MEM}_I$$

[Laws 8 and 32]

$$= \left(\begin{array}{l}(g(b(x)) \mathbin{\&} \Omega_A(A_1));\ \Omega_A(A_2);\ \mathit{terminate} \to \mathit{Skip})\\ \llbracket \emptyset \mid \mathit{MEM}_I \mid \{b\} \rrbracket\\ \mathit{Memory}(b)\end{array}\right) \setminus \mathit{MEM}_I$$

[Ω_A]

$$= \left(\begin{array}{l}(\Omega_A(g(b(x)) \mathbin{\&} A_1);\ \Omega_A(A_2);\ \mathit{terminate} \to \mathit{Skip})\\ \llbracket \emptyset \mid \mathit{MEM}_I \mid \{b\} \rrbracket\\ \mathit{Memory}(b)\end{array}\right) \setminus \mathit{MEM}_I$$

K.15.5 Input Event

$$\left(\left(\begin{array}{l}(\Omega_A(c?y : P(x) \to A_1(y, x));\ \mathit{terminate} \to \mathit{Skip})\\ \llbracket \emptyset \mid \mathit{MEM}_I \mid \{b\} \rrbracket\\ \mathit{Memory}(b)\end{array}\right) \setminus \mathit{MEM}_I\right);$$
$$\left(\left(\begin{array}{l}(\Omega_A(A_2);\ \mathit{terminate} \to \mathit{Skip})\\ \llbracket \emptyset \mid \mathit{MEM}_I \mid \{b\} \rrbracket\\ \mathit{Memory}(b)\end{array}\right) \setminus \mathit{MEM}_I\right)$$
$$= \left(\begin{array}{l}(\Omega_A((c?y : P(y, x) \to A_1(y, x));\ \Omega_A(A_2);\ \mathit{terminate} \to \mathit{Skip})\\ \llbracket \mathit{MEM}_I \rrbracket\\ \mathit{Memory}(b)\end{array}\right) \setminus \mathit{MEM}_I$$

Proof.

$$\left(\left(\begin{array}{l}(\Omega_A(c?y : P(x) \to A_1(y,x));\ \mathit{terminate} \to \mathit{Skip})\\ \lbrack\!\lbrack \emptyset \mid \mathit{MEM}_I \mid \{b\} \rbrack\!\rbrack\\ \mathit{Memory}(b)\end{array}\right) \setminus \mathit{MEM}_I\right);$$
$$\left(\left(\begin{array}{l}(\Omega_A(A_2);\ \mathit{terminate} \to \mathit{Skip})\\ \lbrack\!\lbrack \emptyset \mid \mathit{MEM}_I \mid \{b\} \rbrack\!\rbrack\\ \mathit{Memory}(b)\end{array}\right) \setminus \mathit{MEM}_I\right)$$

$[\Omega_A]$

$$= \left(\left(\begin{array}{l}\left(\begin{array}{l}(get.x?vv_0 \to c?y : P(y,x) \to \Omega_A(A_1(y,x)));\\ \mathit{terminate} \to \mathit{Skip}\end{array}\right)\\ \lbrack\!\lbrack \emptyset \mid \mathit{MEM}_I \mid \{b\} \rbrack\!\rbrack\\ \mathit{Memory}(b)\end{array}\right) \setminus \mathit{MEM}_I\right);$$
$$\left(\left(\begin{array}{l}(\Omega_A(A_2);\ \mathit{terminate} \to \mathit{Skip})\\ \lbrack\!\lbrack \emptyset \mid \mathit{MEM}_I \mid \{b\} \rbrack\!\rbrack\\ \mathit{Memory}(b)\end{array}\right) \setminus \mathit{MEM}_I\right)$$

[Lemma K.5]

$$= \left(\left(\begin{array}{l}((c?y : P(y,b(x)) \to \Omega_A(A_1(y,b(x))));\ \mathit{terminate} \to \mathit{Skip})\\ \lbrack\!\lbrack \emptyset \mid \mathit{MEM}_I \mid \{b\} \rbrack\!\rbrack\\ \mathit{Memory}(b)\end{array}\right) \setminus \mathit{MEM}_I\right);$$
$$\left(\left(\begin{array}{l}(\Omega_A(A_2);\ \mathit{terminate} \to \mathit{Skip})\\ \lbrack\!\lbrack \emptyset \mid \mathit{MEM}_I \mid \{b\} \rbrack\!\rbrack\\ \mathit{Memory}(b)\end{array}\right) \setminus \mathit{MEM}_I\right)$$

[Law 50]

$$= \left(\left(\begin{array}{l}(c?y : P(y,b(x)) \to (\Omega_A(A_1(y,b(x)));\ \mathit{terminate} \to \mathit{Skip}))\\ \lbrack\!\lbrack \emptyset \mid \mathit{MEM}_I \mid \{b\} \rbrack\!\rbrack\\ \mathit{Memory}(b)\end{array}\right) \setminus \mathit{MEM}_I\right);$$
$$\left(\left(\begin{array}{l}(\Omega_A(A_2);\ \mathit{terminate} \to \mathit{Skip})\\ \lbrack\!\lbrack \emptyset \mid \mathit{MEM}_I \mid \{b\} \rbrack\!\rbrack\\ \mathit{Memory}(b)\end{array}\right) \setminus \mathit{MEM}_I\right)$$

[Law 33]

provided

$[c \notin \mathit{MEM}_I]$

$[y \notin \mathit{usedV}(\mathit{Memory}(b))]$

$[\mathit{initials}(\mathit{Memory}(b)) \subseteq \mathit{MEM}_I]$

[$\mathit{Memory}(b)$ is deterministic]

$$= \left(\left(\begin{array}{l} c?y : P(y, b(x)) \to \\ \qquad \left(\begin{array}{l} (\Omega_A(A_1(y, b(x))); \; terminate \to Skip) \\ [\![\emptyset \mid MEM_I \mid \{b\}]\!] \\ Memory(b) \end{array}\right) \end{array}\right) \setminus MEM_I \right); \\ \left(\left(\begin{array}{l} (\Omega_A(A_2); \; terminate \to Skip) \\ [\![\emptyset \mid MEM_I \mid \{b\}]\!] \\ Memory(b) \end{array}\right) \setminus MEM_I \right)$$

[Law 51]

$$= \left(\begin{array}{l} c?y : P(y, b(x)) \to \\ \left(\left(\begin{array}{l} (\Omega_A(A_1(y, b(x))); \; terminate \to Skip) \\ [\![\emptyset \mid MEM_I \mid \{b\}]\!] \\ Memory(b) \end{array}\right) \setminus MEM_I \right) \end{array}\right); \\ \left(\left(\begin{array}{l} (\Omega_A(A_2); \; terminate \to Skip) \\ [\![\emptyset \mid MEM_I \mid \{b\}]\!] \\ Memory(b) \end{array}\right) \setminus MEM_I \right)$$

[Law 50]

provided

$[y \notin FV(A_2) \; by \; renaming \; any \; existing \; y]$

$$= c?y : P(y, b(x)) \to \\ \left(\begin{array}{l} \left(\left(\begin{array}{l} (\Omega_A(A_1(y, b(x))); \; terminate \to Skip) \\ [\![\emptyset \mid MEM_I \mid \{b\}]\!] \\ Memory(b) \end{array}\right) \setminus MEM_I \right); \\ \left(\left(\begin{array}{l} (\Omega_A(A_2); \; terminate \to Skip) \\ [\![\emptyset \mid MEM_I \mid \{b\}]\!] \\ Memory(b) \end{array}\right) \setminus MEM_I \right) \end{array}\right)$$

[IH]

$$= c?y : P(y, b(x)) \to \\ \left(\left(\begin{array}{l} (\Omega_A(A_1(y, b(x))); \; \Omega_A(A_2); \; terminate \to Skip) \\ [\![\emptyset \mid MEM_I \mid \{b\}]\!] \\ Memory(b) \end{array}\right) \setminus MEM_I \right)$$

[Law 51]

$$= \left(\begin{array}{l} c?y : P(y, b(x)) \to \\ \left(\begin{array}{l} (\Omega_A(A_1(y, b(x))); \; \Omega_A(A_2); \; terminate \to Skip) \\ [\![\emptyset \mid MEM_I \mid \{b\}]\!] \\ Memory(b) \end{array}\right) \end{array}\right) \setminus MEM_I$$

[Law 33]

$$= \left(\begin{array}{l} (c?y : P(y, b(x)) \rightarrow (\Omega_A(A_1(y, b(x))); \ \Omega_A(A_2); \ terminate \rightarrow Skip)) \\ \llbracket \emptyset \mid MEM_I \mid \{b\} \rrbracket \\ Memory(b) \end{array}\right) \setminus MEM_I$$

[Law 50]

$$= \left(\begin{array}{l} \left(\begin{array}{l} (c?y : P(y, b(x)) \rightarrow \Omega_A(A_1(y, b(x)))); \\ \Omega_A(A_2); \\ terminate \rightarrow Skip \end{array}\right) \\ \llbracket MEM_I \rrbracket \\ Memory(b) \end{array}\right) \setminus MEM_I$$

[Law 24]

$$= \left(\begin{array}{l} \left(\begin{array}{l} get.x?vv_0 \rightarrow \\ \quad \left(\begin{array}{l} (c?y : P(y, vv_0) \rightarrow \Omega_A(A_1(y, vv_0))); \\ \Omega_A(A_2); \\ terminate \rightarrow Skip \end{array}\right) \end{array}\right) \\ \llbracket MEM_I \rrbracket \\ (get.x!b(x) \rightarrow Memory(b)) \end{array}\right) \setminus MEM_I$$

[Law 10]

$$= \left(\begin{array}{l} \left(\begin{array}{l} get.x?vv_0 \rightarrow \\ \quad \left(\begin{array}{l} (c?y : P(y, vv_0) \rightarrow \Omega_A(A_1(y, vv_0))); \\ \Omega_A(A_2); \\ terminate \rightarrow Skip \end{array}\right) \end{array}\right) \\ \llbracket MEM_I \rrbracket \\ \left(\begin{array}{l} (\Box \, n : NAME \bullet get.n!b(n) \rightarrow Memory(b)) \\ \Box \ (\Box \, n : NAME \bullet set.n?nv \rightarrow Memory(b \oplus \{n \mapsto nv\})) \\ \Box \ terminate \rightarrow Skip \end{array}\right) \end{array}\right) \setminus MEM_I$$

[Lemma K.2]

$$= \left(\begin{array}{l} \left(\begin{array}{l} get.x?vv_0 \rightarrow \\ \quad \left(\begin{array}{l} (c?y : P(y, vv_0) \rightarrow \Omega_A(A_1(y, vv_0))); \\ \Omega_A(A_2); \\ terminate \rightarrow Skip \end{array}\right) \end{array}\right) \\ \llbracket MEM_I \rrbracket \\ Memory(b) \end{array}\right) \setminus MEM_I$$

[Law 50]

$$= \left(\begin{array}{l} \left(\begin{array}{l} (get.x?vv_0 \rightarrow c?y : P(y, vv_0) \rightarrow \Omega_A(A_1(y, vv_0))); \\ \Omega_A(A_2); \\ terminate \rightarrow Skip \end{array}\right) \\ \llbracket MEM_I \rrbracket \\ Memory(b) \end{array}\right) \setminus MEM_I$$

$$[\Omega_A]$$

$$= \left(\begin{array}{l} (\Omega_A(c?y : P(x) \rightarrow A_1(y, x));\ \Omega_A(A_2);\ \mathit{terminate} \rightarrow \mathit{Skip}) \\ \llbracket \mathit{MEM}_I \rrbracket \\ \mathit{Memory}(b) \end{array} \right) \setminus \mathit{MEM}_I$$

K.15.6 Assignment

$$\left(\begin{array}{l} \left(\begin{array}{l} (\Omega_A(x_0, \ldots, x_n := e_0(v_0, \ldots, v_n, l_0, \ldots, l_m), \ldots, e_n(v_0, \ldots, v_n, l_0, \ldots, l_m)); \\ \qquad \mathit{terminate} \rightarrow \mathit{Skip}) \\ \llbracket \emptyset \mid \mathit{MEM}_I \mid \{b\} \rrbracket \\ \mathit{Memory}(b) \end{array} \right) \\ \setminus \mathit{MEM}_I \end{array} \right); \\ \left(\begin{array}{l} \left(\begin{array}{l} (\Omega_A(A_2);\ \mathit{terminate} \rightarrow \mathit{Skip}) \\ \llbracket \emptyset \mid \mathit{MEM}_I \mid \{b\} \rrbracket \\ \mathit{Memory}(b) \end{array} \right) \\ \setminus \mathit{MEM}_I \end{array} \right)$$

$$= \left(\begin{array}{l} (\Omega_A(x_0, \ldots, x_n := e_0(v_0, \ldots, v_n, l_0, \ldots, l_m), \ldots, e_n(v_0, \ldots, v_n, l_0, \ldots, l_m);\ \Omega_A(A_2); \\ \qquad \mathit{terminate} \rightarrow \mathit{Skip}) \\ \llbracket \emptyset \mid \mathit{MEM}_I \mid \{b\} \rrbracket \\ \mathit{Memory}(b) \end{array} \right) \\ \setminus \mathit{MEM}_I$$

Proof. In this proof we will consider a single assignment $x := e(x)$. The generalisation of this proof by induction on the number assigned variables is rather simple, but omitted here for the sake of presentation.

$$\left(\left(\begin{array}{l} (\Omega_A(x := e(x));\ \mathit{terminate} \rightarrow \mathit{Skip}) \\ \llbracket \emptyset \mid \mathit{MEM}_I \mid \{b\} \rrbracket \\ \mathit{Memory}(b) \end{array} \right) \setminus \mathit{MEM}_I \right); \\ \left(\left(\begin{array}{l} (\Omega_A(A_2);\ \mathit{terminate} \rightarrow \mathit{Skip}) \\ \llbracket \emptyset \mid \mathit{MEM}_I \mid \{b\} \rrbracket \\ \mathit{Memory}(b) \end{array} \right) \setminus \mathit{MEM}_I \right)$$

$$[\Omega_A]$$

$$= \left(\left(\begin{array}{l} \left(\begin{array}{l} (\mathit{get}.x?vv_0 \rightarrow \mathit{set}.x!e(vv_0) \rightarrow \mathit{Skip}); \\ \mathit{terminate} \rightarrow \mathit{Skip} \end{array} \right) \\ \llbracket \emptyset \mid \mathit{MEM}_I \mid \{b\} \rrbracket \\ \mathit{Memory}(b) \end{array} \right) \setminus \mathit{MEM}_I \right); \\ \left(\left(\begin{array}{l} (\Omega_A(A_2);\ \mathit{terminate} \rightarrow \mathit{Skip}) \\ \llbracket \emptyset \mid \mathit{MEM}_I \mid \{b\} \rrbracket \\ \mathit{Memory}(b) \end{array} \right) \setminus \mathit{MEM}_I \right)$$

[Lemma K.5]

$$= \left(\left(\begin{array}{l} ((set.x!e(b(x)) \rightarrow \mathit{Skip});\ terminate \rightarrow \mathit{Skip}) \\ \llbracket \emptyset \mid MEM_I \mid \{b\} \rrbracket \\ Memory(b) \end{array} \right) \setminus MEM_I \right); \left(\left(\begin{array}{l} (\Omega_A(A_2);\ terminate \rightarrow \mathit{Skip}) \\ \llbracket \emptyset \mid MEM_I \mid \{b\} \rrbracket \\ Memory(b) \end{array} \right) \setminus MEM_I \right)$$

[Law 34]

$$= \left(\left(\begin{array}{l} (set.x!e(b(x)) \rightarrow terminate \rightarrow \mathit{Skip})) \\ \llbracket \emptyset \mid MEM_I \mid \{b\} \rrbracket \\ Memory(b) \end{array} \right) \setminus MEM_I \right); \left(\left(\begin{array}{l} (\Omega_A(A_2);\ terminate \rightarrow \mathit{Skip}) \\ \llbracket \emptyset \mid MEM_I \mid \{b\} \rrbracket \\ Memory(b) \end{array} \right) \setminus MEM_I \right)$$

[Lemma K.6]

$$= \left(\left(\begin{array}{l} (terminate \rightarrow \mathit{Skip})) \\ \llbracket \emptyset \mid MEM_I \mid \{b\} \rrbracket \\ Memory(b \oplus \{x \mapsto b(x)\}) \end{array} \right) \setminus MEM_I \right); \left(\left(\begin{array}{l} (\Omega_A(A_2);\ terminate \rightarrow \mathit{Skip}) \\ \llbracket \emptyset \mid MEM_I \mid \{b\} \rrbracket \\ Memory(b) \end{array} \right) \setminus MEM_I \right)$$

[Law 49]

$$= \left(\left(\begin{array}{l} (terminate \rightarrow \mathit{Skip})) \\ \llbracket \emptyset \mid MEM_I \mid \{bs\} \rrbracket \\ Memory(bs \oplus \{x \mapsto bs(x)\}) \end{array} \right) \setminus MEM_I \right); \left(\left(\begin{array}{l} (\Omega_A(A_2);\ terminate \rightarrow \mathit{Skip}) \\ \llbracket \emptyset \mid MEM_I \mid \{bs\} \rrbracket \\ Memory(bs) \end{array} \right) \setminus MEM_I \right)$$

[Law 9]

$$
= \left(\begin{array}{l}
\left(\begin{array}{l}
(terminate \rightarrow Skip)) \\
[\![\emptyset \mid MEM_I \mid \{bs\}]\!] \\
\left(\begin{array}{l}
\mathbf{vres}\ b : BINDING \bullet \\
\qquad (\Box\, n : NAME \bullet get.n!b(n) \rightarrow Cell(b)) \\
\qquad \Box\ (\Box\, n : NAME \bullet set.n?nv \rightarrow Cell(b \oplus \{n \mapsto nv\})) \\
\qquad \Box\ terminate \rightarrow Skip
\end{array} \right) \\
\quad (bs \oplus \{x \mapsto bs(x)\})
\end{array} \right) \\
\ \backslash\ MEM_I
\end{array} \right);
$$

$$
\left(\left(\begin{array}{l}
(\Omega_A(A_2);\ terminate \rightarrow Skip) \\
[\![\emptyset \mid MEM_I \mid \{bs\}]\!] \\
Memory(bs)
\end{array} \right) \backslash\ MEM_I \right)
$$

[Semantics of **vres**]

$$
= \left(\begin{array}{l}
\left(\begin{array}{l}
(terminate \rightarrow Skip)) \\
[\![\emptyset \mid MEM_I \mid \{bs\}]\!] \\
\left(\begin{array}{l}
\mathbf{var}\ b : BINDING \bullet \\
\qquad b := bs \oplus \{x \mapsto bs(x)\}; \\
\qquad \left(\begin{array}{l}
(\Box\, n : NAME \bullet get.n!b(n) \rightarrow Cell(b)) \\
\Box\ (\Box\, n : NAME \bullet set.n?nv \rightarrow Cell(b \oplus \{n \mapsto nv\})) \\
\Box\ terminate \rightarrow Skip
\end{array} \right); \\
\qquad bs := b
\end{array} \right)
\end{array} \right) \\
\ \backslash\ MEM_I
\end{array} \right);
$$

$$
\left(\left(\begin{array}{l}
(\Omega_A(A_2);\ terminate \rightarrow Skip) \\
[\![\emptyset \mid MEM_I \mid \{bs\}]\!] \\
Memory(bs)
\end{array} \right) \backslash\ MEM_I \right)
$$

[Law 1]

$$
= \left(\begin{array}{l}
\left(\begin{array}{l}
\mathbf{var}\ b : BINDING \bullet \\
\qquad (terminate \rightarrow Skip)) \\
\qquad [\![\emptyset \mid MEM_I \mid \{bs\}]\!] \\
\qquad \left(\begin{array}{l}
b := bs \oplus \{x \mapsto bs(x)\}; \\
\left(\begin{array}{l}
(\Box\, n : NAME \bullet get.n!b(n) \rightarrow Cell(b)) \\
\Box\ (\Box\, n : NAME \bullet set.n?nv \rightarrow Cell(b \oplus \{n \mapsto nv\})) \\
\Box\ terminate \rightarrow Skip
\end{array} \right); \\
bs := b
\end{array} \right)
\end{array} \right) \\
\ \backslash\ MEM_I
\end{array} \right);
$$

$$
\left(\left(\begin{array}{l}
(\Omega_A(A_2);\ terminate \rightarrow Skip) \\
[\![\emptyset \mid MEM_I \mid \{bs\}]\!] \\
Memory(bs)
\end{array} \right) \backslash\ MEM_I \right)
$$

[Law 47]

$$
= \left(\begin{array}{l}
\left(\begin{array}{l}
\mathbf{var}\ b : \mathit{BINDING} \bullet \\
\quad (\mathit{terminate} \rightarrow \mathit{Skip})) \\
\quad [\![\emptyset \mid \mathit{MEM}_I \mid \{bs\}]\!] \\
\quad \left(\begin{array}{l}
\left(\begin{array}{l}
\left(\begin{array}{l}
\Box\, n : \mathit{NAME} \bullet \\
\quad \mathit{get}.n!(bs \oplus \{x \mapsto bs(x)\})(n) \rightarrow \\
\quad \mathit{Cell}(bs \oplus \{x \mapsto bs(x)\}))
\end{array}\right) \\
\Box \left(\begin{array}{l}
\Box\, n : \mathit{NAME} \bullet \\
\quad \mathit{set}.n?nv \rightarrow \\
\quad \mathit{Cell}((bs \oplus \{x \mapsto bs(x)\}) \oplus \{n \mapsto nv\})
\end{array}\right) \\
\Box\ \mathit{terminate} \rightarrow \mathit{Skip}
\end{array}\right); \\
bs := bs \oplus \{x \mapsto bs(x)\}
\end{array}\right)
\end{array}\right) \\
\quad \setminus \mathit{MEM}_I
\end{array}\right);
$$

$$
\left(\left(\begin{array}{l}
(\Omega_A(A_2);\ \mathit{terminate} \rightarrow \mathit{Skip}) \\
[\![\emptyset \mid \mathit{MEM}_I \mid \{bs\}]\!] \\
\mathit{Memory}(bs)
\end{array}\right) \setminus \mathit{MEM}_I\right)
$$

[Laws 6]

$$
= \left(\begin{array}{l}
\left(\begin{array}{l}
(\mathit{terminate} \rightarrow \mathit{Skip})) \\
[\![\emptyset \mid \mathit{MEM}_I \mid \{bs\}]\!] \\
\left(\begin{array}{l}
\left(\begin{array}{l}
\left(\begin{array}{l}
\Box\, n : \mathit{NAME} \bullet \\
\quad \mathit{get}.n!(bs \oplus \{x \mapsto bs(x)\})(n) \rightarrow \\
\quad \mathit{Cell}(bs \oplus \{x \mapsto bs(x)\}))
\end{array}\right) \\
\Box \left(\begin{array}{l}
\Box\, n : \mathit{NAME} \bullet \\
\quad \mathit{set}.n?nv \rightarrow \\
\quad \mathit{Cell}((bs \oplus \{x \mapsto bs(x)\}) \oplus \{n \mapsto nv\})
\end{array}\right) \\
\Box\ \mathit{terminate} \rightarrow \mathit{Skip}
\end{array}\right); \\
bs := bs \oplus \{x \mapsto bs(x)\}
\end{array}\right)
\end{array}\right) \\
\quad \setminus \mathit{MEM}_I
\end{array}\right);
$$

$$
\left(\left(\begin{array}{l}
(\Omega_A(A_2);\ \mathit{terminate} \rightarrow \mathit{Skip}) \\
[\![\emptyset \mid \mathit{MEM}_I \mid \{bs\}]\!] \\
\mathit{Memory}(bs)
\end{array}\right) \setminus \mathit{MEM}_I\right)
$$

[Law 49]

$$
= \left(\begin{array}{l}
\left(\begin{array}{l}
(terminate \rightarrow Skip)) \\
[\![\emptyset \mid MEM_I \mid \{bs\}]\!] \\
\left(\begin{array}{l}
\left(\begin{array}{l}
\left(\begin{array}{l}
\Box\, n : NAME \bullet \\
\qquad get.n!(b \oplus \{x \mapsto b(x)\})(n) \rightarrow \\
\qquad Cell(b \oplus \{x \mapsto b(x)\}))
\end{array}\right) \\
\Box \left(\begin{array}{l}
\Box\, n : NAME \bullet \\
\qquad set.n?nv \rightarrow \\
\qquad Cell((s \oplus \{x \mapsto s(x)\}) \oplus \{n \mapsto nv\})
\end{array}\right) \\
\Box\, terminate \rightarrow Skip
\end{array}\right) ; \\
b := b \oplus \{x \mapsto b(x)\}
\end{array}\right)
\end{array}\right) \\
\quad \setminus MEM_I
\end{array}\right) ;
$$

$$
\left(\left(\begin{array}{l}
(\Omega_A(A_2);\ terminate \rightarrow Skip) \\
[\![\emptyset \mid MEM_I \mid \{b\}]\!] \\
Memory(b)
\end{array}\right) \setminus MEM_I \right)
$$

[Laws 36 and 37]

$$
= \left(\begin{array}{l}
\left(\begin{array}{l}
(terminate \rightarrow Skip)) \\
[\![\emptyset \mid MEM_I \mid \{bs\}]\!] \\
\left(\left(\begin{array}{l}
\left(\begin{array}{l}
\Box\, n : NAME \bullet \\
\qquad get.n!(b \oplus \{x \mapsto b(x)\})(n) \rightarrow \\
\qquad Cell(b \oplus \{x \mapsto b(x)\})); \\
\qquad b := b \oplus \{x \mapsto b(x)\}
\end{array}\right) \\
\Box \left(\begin{array}{l}
\Box\, n : NAME \bullet \\
\qquad set.n?nv \rightarrow \\
\qquad Cell((s \oplus \{x \mapsto s(x)\}) \oplus \{n \mapsto nv\}); \\
\qquad b := b \oplus \{x \mapsto b(x)\}
\end{array}\right) \\
\Box\, terminate \rightarrow Skip;\ b := b \oplus \{x \mapsto b(x)\}
\end{array}\right)\right)
\end{array}\right) \\
\quad \setminus MEM_I
\end{array}\right) ;
$$

$$
\left(\left(\begin{array}{l}
(\Omega_A(A_2);\ terminate \rightarrow Skip) \\
[\![\emptyset \mid MEM_I \mid \{b\}]\!] \\
Memory(b)
\end{array}\right) \setminus MEM_I \right)
$$

[Law 10]

$$
= \left(\begin{array}{l}
\left(\begin{array}{l}
(terminate \rightarrow Skip) \\
[\![\emptyset \mid MEM_I \mid \{bs\}]\!] \\
(terminate \rightarrow Skip;\ b := b \oplus \{x \mapsto b(x)\})
\end{array}\right) \\
\quad \setminus MEM_I
\end{array}\right) ;
$$

$$
\left(\left(\begin{array}{l}
(\Omega_A(A_2);\ terminate \rightarrow Skip) \\
[\![\emptyset \mid MEM_I \mid \{b\}]\!] \\
Memory(b)
\end{array}\right) \setminus MEM_I \right)
$$

[Law 34]

$$= \left(\begin{array}{l}\left(\begin{array}{l}(\mathit{terminate} \to \mathit{Skip})\\ [\![\emptyset \mid \mathit{MEM}_I \mid \{bs\}]\!]\\ (\mathit{terminate} \to b := b \oplus \{x \mapsto b(x)\})\end{array}\right)\\ \quad \setminus \mathit{MEM}_I\end{array}\right);$$
$$\left(\left(\begin{array}{l}(\Omega_A(A_2);\ \mathit{terminate} \to \mathit{Skip})\\ [\![\emptyset \mid \mathit{MEM}_I \mid \{b\}]\!]\\ \mathit{Memory}(b)\end{array}\right) \setminus \mathit{MEM}_I\right)$$

[Law 25]

$$= \left(\left(\begin{array}{l}\mathit{Skip}\\ [\![\emptyset \mid \mathit{MEM}_I \mid \{bs\}]\!]\\ b := b \oplus \{x \mapsto b(x)\}\end{array}\right) \setminus \mathit{MEM}_I\right);$$
$$\left(\left(\begin{array}{l}(\Omega_A(A_2);\ \mathit{terminate} \to \mathit{Skip})\\ [\![\emptyset \mid \mathit{MEM}_I \mid \{b\}]\!]\\ \mathit{Memory}(b)\end{array}\right) \setminus \mathit{MEM}_I\right)$$

[Law 8 and 30]

$$= \left(\left(\begin{array}{l}b := b \oplus \{x \mapsto b(x)\};\\ \left(\begin{array}{l}\mathit{Skip}\\ [\![\emptyset \mid \mathit{MEM}_I \mid \{bs\}]\!]\\ \mathit{Skip}\end{array}\right)\end{array}\right) \setminus \mathit{MEM}_I\right);$$
$$\left(\left(\begin{array}{l}(\Omega_A(A_2);\ \mathit{terminate} \to \mathit{Skip})\\ [\![\emptyset \mid \mathit{MEM}_I \mid \{b\}]\!]\\ \mathit{Memory}(b)\end{array}\right) \setminus \mathit{MEM}_I\right)$$

[Laws 28 and 8]

$$= ((b := b \oplus \{x \mapsto b(x)\}) \setminus \mathit{MEM}_I);$$
$$\left(\left(\begin{array}{l}(\Omega_A(A_2);\ \mathit{terminate} \to \mathit{Skip})\\ [\![\emptyset \mid \mathit{MEM}_I \mid \{b\}]\!]\\ \mathit{Memory}(b)\end{array}\right) \setminus \mathit{MEM}_I\right)$$

[Law 31]

$$= \left(\begin{array}{l}(b := b \oplus \{x \mapsto b(x)\});\\ \left(\begin{array}{l}(\Omega_A(A_2);\ \mathit{terminate} \to \mathit{Skip})\\ [\![\emptyset \mid \mathit{MEM}_I \mid \{b\}]\!]\\ \mathit{Memory}(b)\end{array}\right)\end{array}\right) \setminus \mathit{MEM}_I$$

[Law 47]

$$= \left(\begin{array}{l}(\Omega_A(A_2);\ \mathit{terminate} \to \mathit{Skip})\\ [\![\emptyset \mid \mathit{MEM}_I \mid \{b\}]\!]\\ \mathit{Memory}(b \oplus \{x \mapsto b(x)\})\end{array}\right) \setminus \mathit{MEM}_I$$

[Lemma K.6]

$$= \left(\begin{array}{l} (set.x!e(b(x)) \to \Omega_A(A_2);\ terminate \to Skip) \\ \llbracket \emptyset \mid MEM_I \mid \{b\} \rrbracket \\ Memory(b) \end{array}\right) \setminus MEM_I$$

[Law 34]

$$= \left(\begin{array}{l} ((set.x!e(b(x)) \to Skip);\ \Omega_A(A_2);\ terminate \to Skip) \\ \llbracket \emptyset \mid MEM_I \mid \{b\} \rrbracket \\ Memory(b) \end{array}\right) \setminus MEM_I$$

[Lemma K.5]

$$= \left(\begin{array}{l} ((get.x?vv_0 \to set.x!e(vv_0) \to Skip);\ \Omega_A(A_2);\ terminate \to Skip) \\ \llbracket \emptyset \mid MEM_I \mid \{b\} \rrbracket \\ Memory(b) \end{array}\right) \setminus MEM_I$$

[Ω_A]

$$= \left(\begin{array}{l} (\Omega_A(x := e(x));\ \Omega_A(A_2);\ terminate \to Skip) \\ \llbracket \emptyset \mid MEM_I \mid \{b\} \rrbracket \\ Memory(b) \end{array}\right) \setminus MEM_I$$

□

Lemma K.8

$$\begin{array}{l} P. \\ \quad \mathbf{var}\ b : \{x : BINDING \mid b(v_0) \in T_0 \wedge inv(b(v_0))\} \bullet \\ \quad\quad \left(\begin{array}{l} \left(\begin{array}{l} \mathbf{if}\ g_0(b(v_0)) \to\ \Omega_A(A_0) \\ \quad \talloblong\ \ldots \\ \quad \talloblong\ g_n(b(v_0)) \to\ \Omega_A(A_n) \\ \mathbf{fi} \end{array}\right); \\ terminate \to Skip \\ \llbracket \emptyset \mid MEM_I \mid \{b\} \rrbracket \\ Memory(b) \end{array}\right) \\ \quad\quad \setminus MEM_I \\ = \\ P_S(Chaos) \end{array}$$

provided $\bigvee i \bullet g_i \equiv false$

Proof.

$P.$

$$\begin{array}{l}\textbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \wedge \mathit{inv}(b(v_0))\} \bullet \\ \quad \left(\begin{array}{l}\left(\begin{array}{l}\textbf{if}\ g_0(b(v_0)) \rightarrow\ \Omega_A(A_0) \\ \quad [\!]\ \ldots \\ \quad [\!]\ g_n(v_0) \rightarrow\ \Omega_A(A_n) \\ \textbf{fi}\end{array}\right); \\ \mathit{terminate} \rightarrow \mathit{Skip} \\ [\![\emptyset \mid \mathit{MEM}_I \mid \{b\}]\!] \\ \mathit{Memory}(b)\end{array}\right) \\ \quad \setminus \mathit{MEM}_I\end{array}$$

[Assuming that no alternative is true Law 54]

$$= P.$$

$$\begin{array}{l}\textbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \wedge \mathit{inv}(b(v_0))\} \bullet \\ \quad \left(\begin{array}{l}\mathit{Chaos}; \mathit{terminate} \rightarrow \mathit{Skip} \\ [\![\emptyset \mid \mathit{MEM}_I \mid \{b\}]\!] \\ \mathit{Memory}(b)\end{array}\right) \setminus \mathit{MEM}_I\end{array}$$

[Law 40]

$$= P.$$

$$\begin{array}{l}\textbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \wedge \mathit{inv}(b(v_0))\} \bullet \\ \quad \left(\begin{array}{l}\mathit{Chaos} \\ [\![\emptyset \mid \mathit{MEM}_I \mid \{b\}]\!] \\ \mathit{Memory}(b)\end{array}\right) \setminus \mathit{MEM}_I\end{array}$$

[Law 41]

$$= P.$$

$$\begin{array}{l}\textbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \wedge \mathit{inv}(b(v_0))\} \bullet \\ \quad \mathit{Chaos} \setminus \mathit{MEM}_I\end{array}$$

[Law 39]

$$= P.$$

$$\begin{array}{l}\textbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \wedge \mathit{inv}(b(v_0))\} \bullet \\ \quad \mathit{Chaos}\end{array}$$

[Law 26(b is the only component of S)]

$$= P.\mathit{Chaos}$$

[Law 54]

provided

$\bigvee i \bullet g_i \equiv \mathit{false}$

$$= P_S . \left(\begin{array}{l} \textbf{if } g_0(v_0) \rightarrow \ A_0 \\ \quad [\!] \ \ldots \\ \quad [\!] \ g_n(v_0) \rightarrow \ A_n \\ \textbf{fi} \end{array} \right)$$

□

Lemma K.9

$$\begin{array}{l} P . \\ \quad \textbf{var } b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \wedge \mathit{inv}(b(v_0))\} \bullet \\ \quad \left(\begin{array}{l} \left(\begin{array}{l} \textbf{if } g_0(b(v_0)) \rightarrow \ \Omega_A(A_0) \\ \quad [\!] \ \ldots \\ \quad [\!] \ g_n(b(v_0)) \rightarrow \ \Omega_A(A_n) \\ \textbf{fi} \end{array} \right) ; \\ \mathit{terminate} \rightarrow \mathit{Skip} \\ [\![\emptyset \mid \mathit{MEM}_I \mid \{b\}]\!] \\ \mathit{Memory}(b) \end{array} \right) \\ \quad \setminus \mathit{MEM}_I \\ = \\ P_S(A_i) \end{array}$$

provided $g_i \equiv \mathit{true} \ \mathit{and} \bigvee j : \{0, \ldots, n\} \setminus \{i\} \bullet g_j \equiv \mathit{false}$

Proof.

$$\begin{array}{l} = P . \\ \quad \textbf{var } b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \wedge \mathit{inv}(b(v_0))\} \bullet \\ \quad \left(\begin{array}{l} \left(\begin{array}{l} \textbf{if } g_0(b(v_0)) \rightarrow \ \Omega_A(A_0) \\ \quad [\!] \ \ldots \\ \quad [\!] \ g_n(b(v_0)) \rightarrow \ \Omega_A(A_n) \\ \textbf{fi} \end{array} \right) ; \\ \mathit{terminate} \rightarrow \mathit{Skip} \\ [\![\emptyset \mid \mathit{MEM}_I \mid \{b\}]\!] \\ \mathit{Memory}(b) \end{array} \right) \\ \quad \setminus \mathit{MEM}_I \end{array}$$

[Law 55 (alternative i is true)]

$= P.$

$$\mathbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \land \mathit{inv}(b(v_0))\} \bullet \left(\begin{array}{l} \Omega_A(A_i);\mathit{terminate} \to \mathit{Skip} \\ [\![\emptyset \mid \mathit{MEM}_I \mid \{b\}]\!] \\ \mathit{Memory}(b) \end{array}\right) \setminus \mathit{MEM}_I$$

[IH]

$$= P.\ \mathbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \land \mathit{inv}(b(v_0))\} \bullet \big(\ (\mathbf{vres}\ x : \mathit{BINDING} \bullet A_i(x))(b)\ \big)$$

[Semantics]

$$= P.\ \mathbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \land \mathit{inv}(b(v_0))\} \bullet \big(\ (\mathbf{var}\ x : \mathit{BINDING} \bullet x := b;\ A_i(x);\ b := x)\ \big)$$

[Law 47]

$$= P.\ \mathbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \land \mathit{inv}(b(v_0))\} \bullet \big(\ (\mathbf{var}\ x : \mathit{BINDING} \bullet A_i(b);\ b := b)\ \big)$$

[Laws 48 and 8]

$$= P.\ \mathbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \land \mathit{inv}(b(v_0))\} \bullet \big(\ \mathbf{var}\ x : \mathit{BINDING} \bullet A_i(b)\ \big)$$

[Law 6]

provided

$[x \notin FV(A_i(b))]$

$$= P.\ \mathbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \land \mathit{inv}(b(v_0))\} \bullet A_i(b)$$

[Law 26(b is the only component of S)]

$$= P.(A_i)$$

[Law 55]

provided

$g_i \equiv \mathit{true}$ and $\bigvee j : \{0, \ldots, n\} \setminus \{i\} \bullet g_j \equiv \mathit{false}$

$$= P_S.\left(\begin{array}{l}\textbf{if } g_0(v_0) \rightarrow\ A_0\\ \quad [\!]\ \ldots\\ \quad [\!]\ g_n(v_0) \rightarrow\ A_n\\ \textbf{fi}\end{array}\right)$$

□

Lemma K.10

$$\begin{array}{l}P.\\ \quad \textbf{var } b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \wedge \mathit{inv}(b(v_0))\} \bullet\\ \qquad \left(\begin{array}{l}\left(\begin{array}{l}\textbf{if } g_0(b(v_0)) \rightarrow\ \Omega_A(A_0)\\ \quad [\!]\ \ldots\\ \quad [\!]\ g_n(b(v_0)) \rightarrow\ \Omega_A(A_n)\\ \textbf{fi}\end{array}\right);\\ \mathit{terminate} \rightarrow \mathit{Skip}\\ [\![\emptyset \mid \mathit{MEM}_I \mid \{b\}]\!]\\ \mathit{Memory}(b)\end{array}\right)\\ \qquad \setminus \mathit{MEM}_I\\ =\\ Ps.(A_i \sqcap A_j)\end{array}$$

provided $g_i \wedge g_j \equiv \mathit{true}\ \ \mathit{and} \bigvee x : \{0, \ldots, n\} \setminus \{i, j\} \bullet g_x \equiv \mathit{false}$

Proof. For simplicity, we assume two guards, i and j, are true.

$$\begin{array}{l}= P.\\ \quad \textbf{var } b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \wedge \mathit{inv}(b(v_0))\} \bullet\\ \qquad \left(\begin{array}{l}\left(\begin{array}{l}\textbf{if } g_0(b(v_0)) \rightarrow\ \Omega_A(A_i)\\ \quad [\!]\ \ldots\\ \quad [\!]\ g_n(b(v_0)) \rightarrow\ \Omega_A(A_n)\\ \textbf{fi}\end{array}\right);\\ \mathit{terminate} \rightarrow \mathit{Skip}\\ [\![\emptyset \mid \mathit{MEM}_I \mid \{b\}]\!]\\ \mathit{Memory}(b)\end{array}\right)\\ \qquad \setminus \mathit{MEM}_I\end{array}$$

[Assuming that some alternatives are true Law 55]

$= P.$

$$\begin{array}{l}\textbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \land \mathit{inv}(b(v_0))\} \bullet \\ \left(\begin{array}{l}\Omega_A(A_i) \sqcap \Omega_A(A_j);\\ \mathit{terminate} \to \mathit{Skip}\\ [\![\emptyset \mid \mathit{MEM}_I \mid \{b\}]\!]\\ \mathit{Memory}(b)\end{array}\right) \setminus \mathit{MEM}_I\end{array}$$

[Law 43]

$$= P.$$

$$\begin{array}{l}\textbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \land \mathit{inv}(b(v_0))\} \bullet \\ \left(\begin{array}{l}\left(\begin{array}{l}((\Omega_A(A_i);\mathit{terminate} \to \mathit{Skip})\\ \sqcap (\Omega_A(A_j);\mathit{terminate} \to \mathit{Skip}))\end{array}\right)\\ [\![\emptyset \mid \mathit{MEM}_I \mid \{b\}]\!]\\ \mathit{Memory}(b))\end{array}\right) \setminus \mathit{MEM}_I\end{array}$$

[Law 42]

$$= P.$$

$$\begin{array}{l}\textbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \land \mathit{inv}(b(v_0))\} \bullet \\ \left(\begin{array}{l}(\Omega_A(A_i);\mathit{terminate} \to \mathit{Skip}\\ [\![\emptyset \mid \mathit{MEM}_I \mid \{b\}]\!]\\ \mathit{Memory}(b))\\ \sqcap\\ (\Omega_A(A_j);\mathit{terminate} \to \mathit{Skip}\\ [\![\emptyset \mid \mathit{MEM}_I \mid \{b\}]\!]\\ \mathit{Memory}(b))\end{array}\right) \setminus \mathit{MEM}_I\end{array}$$

[Law 53]

$$= P.$$

$$\begin{array}{l}\textbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \land \mathit{inv}(b(v_0))\} \bullet \\ \left(\begin{array}{l}\left(\begin{array}{l}(\Omega_A(A_i);\mathit{terminate} \to \mathit{Skip}\\ [\![\emptyset \mid \mathit{MEM}_I \mid \{b\}]\!]\\ \mathit{Memory}(b))\end{array}\right) \setminus \mathit{MEM}_I\\ \sqcap\\ \left(\begin{array}{l}(\Omega_A(A_j);\mathit{terminate} \to \mathit{Skip}\\ [\![\emptyset \mid \mathit{MEM}_I \mid \{b\}]\!]\\ \mathit{Memory}(b))\end{array}\right) \setminus \mathit{MEM}_I\end{array}\right)\end{array}$$

[IH]

$$= P.$$

$$\begin{array}{l}\textbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \land \mathit{inv}(b(v_0))\} \bullet \\ \left(\begin{array}{l}(\textbf{vres}\ x : \mathit{BINDING} \bullet A_i(x))(b) \sqcap\\ (\textbf{vres}\ x : \mathit{BINDING} \bullet A_j(x))(b)\end{array}\right)\end{array}$$

[Semantics]

$= P.$

$$\mathbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \land \mathit{inv}(b(v_0))\} \bullet \left(\begin{array}{l}(\mathbf{var}\ x : \mathit{BINDING} \bullet x := b;\ A_i(x);\ b := x) \sqcap \\ (\mathbf{var}\ x : \mathit{BINDING} \bullet x := b;\ A_j(x);\ b := x)\end{array}\right)$$

[Law 49]

$= P.$

$$\mathbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \land \mathit{inv}(b(v_0))\} \bullet \left(\begin{array}{l}(\mathbf{var}\ x : \mathit{BINDING} \bullet x := b;\ A_i(x);\ b := x) \sqcap \\ (\mathbf{var}\ y : \mathit{BINDING} \bullet y := b;\ A_j(y);\ b := y)\end{array}\right)$$

[Law 47]

$= P.$

$$\mathbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \land \mathit{inv}(b(v_0))\} \bullet \left(\begin{array}{l}(\mathbf{var}\ x : \mathit{BINDING} \bullet x := b;\ A_i(x);\ b := x) \sqcap \\ (\mathbf{var}\ y : \mathit{BINDING} \bullet A_j(b);\ b := b)\end{array}\right)$$

[Laws 48 and 8]

$= P.$

$$\mathbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \land \mathit{inv}(b(v_0))\} \bullet \left(\begin{array}{l}(\mathbf{var}\ x : \mathit{BINDING} \bullet x := b;\ A_i(x);\ b := x) \sqcap \\ (\mathbf{var}\ y : \mathit{BINDING} \bullet A_j(b))\end{array}\right)$$

[Law 47]

$= P.$

$$\mathbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \land \mathit{inv}(b(v_0))\} \bullet \left(\begin{array}{l}(\mathbf{var}\ x : \mathit{BINDING} \bullet A_i(b);\ b := b) \sqcap \\ (\mathbf{var}\ y : \mathit{BINDING} \bullet A_j(b))\end{array}\right)$$

[Laws 48 and 8]

$= P.$

$$\mathbf{var}\ b : \{x : \mathit{BINDING} \mid b(v_0) \in T_0 \land \mathit{inv}(b(v_0))\} \bullet \left(\begin{array}{l}(\mathbf{var}\ x : \mathit{BINDING} \bullet A_i(b)) \sqcap \\ (\mathbf{var}\ y : \mathit{BINDING} \bullet A_j(b))\end{array}\right)$$

[Law 6]

provided

$[x \notin FV(A_2(b))]$

$= P.$

$$\begin{array}{l}\textbf{var } b : \{x : BINDING \mid b(v_0) \in T_0 \wedge inv(b(v_0))\} \bullet \\ \quad \left(\left(\textbf{ var } x : BINDING \bullet A_i(b) \right) \sqcap A_j(b) \right)\end{array}$$

[Law 6]

provided

$[x \notin FV(A_i(b))]$

$= P.$

$$\begin{array}{l}\textbf{var } b : \{x : BINDING \mid b(v_0) \in T_0 \wedge inv(b(v_0))\} \bullet \\ \quad \left(A_i(b) \sqcap A_j(b) \right)\end{array}$$

[Law 26(b is the only component of S)]

$=$

$P.(A_i \sqcap \ A_j)$

[Law 55]

provided

$g_i \equiv true$ and $\bigvee j : \{0, \dots, n\} \setminus \{i\} \bullet g_j \equiv false$

$$= P_S. \left(\begin{array}{l} \textbf{if } g_0(v_0) \rightarrow \ A_0 \\ \quad [\!] \ \dots \\ \quad [\!] \ g_n(v_0) \rightarrow \ A_n \\ \textbf{fi} \end{array} \right)$$

$\square$

References

[AB03] A. Aldini and M. Bernardo. A general approach to deadlock freedom verification for software architectures. In *International Symposium of Formal Methods Europe*, volume 2805 of *Lecture Notes in Computer Science*, pages 658–677. Springer, 2003.

[ACN02] J. Aldrich, C. Chambers, and D. Notkin. Archjava: connecting software architecture to implementation. In *International Conference on Software Engineering.* ACM Press, 2002.

[ADG98] R. Allen, R. Douence, and D. Garlan. Specifying and analyzing dynamic software architectures. In *Conference on Fundamental Approaches to Software Engineering (FASE)*, Lisbon, Portugal, March 1998.

[All97a] R. Allen. *A Formal Approach to Software Architecture.* PhD thesis, Carnegie Mellon University, 1997. Issued as CMU Technical Report CMUU-CS–97–144.

[All97b] R. Allen. *A Formal Approach to Software Architecture.* PhD thesis, Carnegie Mellon University, 1997. CMU Technical Report CMUU-CS–97–144.

[Arb04] F. Arbab. Reo: a channel-based coordination model for component composition. *Mathematical. Structures in Computer Science*, 14(3):329–366, 2004.

[BBC+12] Victor Bandur, Jeremy Bryans, Ana Cavalcanti, Andy Galloway, and Jim Woodcock. CML Definition 1. Technical Report D23.2, COMPASS Deliverable, September 2012.

[BBT01] A. Bracciali, A. Brogi, and F. Turini. Coordinating interaction patterns. In *ACM Symposium on Applied Computing*, pages 159–165. ACM, 2001.

[BCD02] M. Bernardo, P. Ciancarini, and L. Donatiello. Architecting families of software systems with process algebras. *ACM Transactions on Software Engineering and Methodology*, 11(4):386–426, 2002.

[BCL+06] E. Bruneton, T. Coupaye, M. Leclercq, V. Quéma, and J.B. Stefani. The FRACTAL component model and its support in Java. *Software: Practice and Experience*, 36(11-12):1257–1284, 2006.

[BGL$^+$08] A. Basu, M. Gallien, C. Lesire, T.-H. Nguyen, S. Bensalem, F. Ingrand, and J. Sifakis. Incremental component-based construction and verification of a robotic system. In *18th European Conference on Artificial Intelligence*, volume 178 of *Frontiers in Artificial Intelligence and Applications*, pages 631–635. IOS Press, 2008.

[BHP06] T. Bures, P. Hnetynka, and F. Plasil. SOFA 2.0: Balancing Advanced Features in a Hierarchical Component Model. In *4th International Conference on Software Engineering Research, Management and Applications*, pages 40–48. IEEE, 2006.

[Cav97] A. L. C. Cavalcanti. *A Refinement Calculus for Z*. PhD thesis, Oxford University Computing Laboratory, Oxford, 1997. Technical Monograph TM-PRG-123, ISBN 00902928-97-X.

[CCH$^+$09] E. Cheung, X. Chen, H. Hsieh, A. Davare, A. Sangiovanni-Vincentelli, and Y. Watanabe. Runtime deadlock analysis for system level design. *Design Automation for Embedded Systems*, 13(4):287–310, 2009.

[CCO11] A. L. C. Cavalcanti, P. Clayton, and C. O'Halloran. From Control Law Diagrams to Ada via *Circus*. *Formal Aspects of Computing*, 23(4):465 – 512, 2011.

[CG07] A. L. C. Cavalcanti and M.-C. Gaudel. Testing for Refinement in CSP. In *9th International Conference on Formal Engineering Methods*, volume 4789 of *Lecture Notes in Computer Science*, pages 151 – 170. Springer-Verlag, 2007.

[CG10] A. L. C. Cavalcanti and M.-C. Gaudel. A note on traces refinement and the *conf* relation in the Unifying Theories of Programming. In A. Butterfield, editor, *Unifying Theories of Programming 2008*, volume 5713 of *Lecture Notes in Computer Science*. Springer-Verlag, 2010.

[Chi09] Z. Chi. Components Composition Compatibility Checking Based on Behavior Description and Roles Division. In *International Conference on Management of e-Commerce and e-Government*, pages 262–265. IEEE, 2009.

[CHLZ07] X. Chen, J. He, Z. Liu, and N. Zhan. A model of Component-Based programming. In *International Symposium on Fundamentals of Software Engineering*, volume 4767 of *Lecture Notes in Computer Science*, pages 191–206. Springer, 2007.

[CK96] S. Cheung and J. Kramer. Context constraints for compositional reachability analysis. *ACM Transactions on Software Engineering and Methodology*, 5(4):334–377, 1996.

[CML+12] Joey W. Coleman, Anders Kaels Malmos, Peter Gorm Larsen, Jan Peleska, Ralph Hains, Zoe Andrews, Richard Payne, Simon Foster, Alvaro Miyazawa, Cristiano Bertolini, and André Didier. COMPASS Tool Vision for a System of Systems Collaborative Development Environment. In *Proceedings of the 7th International Conference on System of System Engineering, IEEE SoSE 2012*, volume 6 of *IEEE Systems Journal*, July 2012.

[CSW03] A. L. C. Cavalcanti, A. C. A. Sampaio, and J. C. P. Woodcock. A Refinement Strategy for ***Circus***. *Formal Aspects of Computing*, **15**(2–3):146–181, 2003.

[CSW05a] A. L. C. Cavalcanti, A. Sampaio, and J. C. P. Woodcock. Unifying classes and processes. *Journal of Software and Systems Modeling*, **4**(3):277–296, 2005.

[CSW05b] A. L. C. Cavalcanti, A. C. A. Sampaio, and J. C. P. Woodcock. Unifying Classes and Processes. *Software and System Modelling*, **4**(3):277–296, 2005.

[CW04] A. L. C. Cavalcanti and J. C. P. Woodcock. A tutorial introduction to CSP in Unifying Theories of Programming. In *Proceedings of the Pernambuco Summer School on Software Engineering: Refinement*. Springer-Verlag, December 2004.

[CW06] A. L. C. Cavalcanti and J. C. P. Woodcock. A Tutorial Introduction to CSP in Unifying Theories of Programming. In A. L. C. Cavalcanti, A. C. A. Sampaio, and J. C. P. Woodcock, editors, *Refinement Techniques in Software Engineering*, volume **3167** of *Lecture Notes in Computer Science*, pages 220 – 268. Springer-Verlag, 2006.

[CZ07] D.C. Craig and WM Zuberek. Compatibility of software components-modeling and verification. In *International Conference on Dependability of Computer Systems*, pages 11–18. IEEE, 2007.

[Dij76] E. W. Dijkstra. *A Discipline of Programming*. Prentice-Hall, 1976.

[DK06] L. DeMichiel and M. Keith. Enterprise javabeans specification, version 3.0. Technical Report JSR 220, Sun Microsystems, 2006.

[DR02] MS Dias and DJ Richardson. Identifying cause and effect relations between events in concurrent event-based components. In *17th IEEE International Conference on Automated Software Engineering*, pages 245–248. IEEE, 2002.

[DZL10] J. Ding, H. Zhu, and Q. Li. Formal Modeling and Verifications of Deadlock Prevention Solutions in Web Service Oriented System. In *2010 17th IEEE International Conference and Workshops on Engineering of Computer-Based Systems*, pages 335–343. IEEE, 2010.

[FG03] A. Farias and Y. Guéhéneuc. On the coherence of component protocols. *Electronic Notes Theoretical Computer Science*, 82(5):42–53, 2003.

[FL09] John Fitzgerald and Peter Gorm Larsen. *Modelling Systems: Practical Tools and Techniques in Software Development*. Cambridge University Press, 2nd edition, 2009.

[FLF01] R.B. Findler, M. Latendresse, and M. Felleisen. Behavioral contracts and behavioral subtyping. *ACM SIGSOFT Software Engineering Notes*, 26(5):229–236, 2001.

[FMS08] Sanford Friedenthal, Alan Moore, and Rick Steiner. *A Practical Guide to SysML: Systems Modeling Language*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2008.

[GGMC$^+$06] G. Gößler, S. Graf, M. Majster-Cederbaum, M. Martens, and J. Sifakis. Ensuring properties of interaction systems. In *Theory and Practice on Program Analysis and Compilation*, volume 4444 of *Lecture Notes in Computer Science*, pages 201–224. Springer, 2006.

[GGMC$^+$07] G. Gößler, S. Graf, M. Majster-Cederbaum, M. Martens, and J. Sifakis. An approach to modelling and verification of component based systems. In *Current Trends in Theory and Practice of Computer Science*, volume 4362 of *Lecture Notes in Computer Science*, pages 295–308. Springer, 2007.

[HGK^{+}06] M. Hepner, R. Gamble, M. Kelkar, L. Davis, and D. Flagg. Patterns of conflict among software components. *The Journal of Systems & Software*, 79(4):537–551, 2006.

[HJ98] C. A. R. Hoare and H. Jifeng. *Unifying Theories of Programming*. Prentice-Hall, 1998.

[HJK10a] R. Hennicker, S. Janisch, and A. Knapp. On the observable behaviour of composite components. *Electronic Notes in Theoretical Computer Science*, 260:125–153, 2010.

[HJK10b] Rolf Hennicker, Stephan Janisch, and Alexander Knapp. On the observable behaviour of composite components. *ENTCS*, 260:125–153, 2010.

[HLL06a] J. He, X. Li, and Z. Liu. rCOS: a refinement calculus of object systems. *Theoretical Computer Science*, 365(1-2):109–142, 2006.

[HLL06b] J. He, X. Li, and Z. Liu. A theory of reactive components. *Electronic Notes in Theoretical Computer Science*, 160:173–195, 2006.

[Hoa85] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.

[IM08] J. Ivers and G. Moreno. PACC starter kit: developing software with predictable behavior. In *ICSE Companion*, pages 949–950. ACM, 2008.

[IR08] Y. Isobe and M. Roggenbach. CSP-Prover – a Proof Tool for the Verification of Scalable Concurrent Systems. *Journal of Computer Software*, 25(4):85 – 92, 2008.

[Jon90] C. B. Jones. *Systematic Software Development Using VDM*. Prentice-Hall, 2nd edition, 1990.

[Kwi07] X.W.M. Kwiatkowska. Compositional state space reduction using untangled actions. In *13th International Workshop on Expressiveness in Concurrency*, volume 175 of *Electronic Notes in Theoretical Computer Science*, pages 27–46, 2007.

[Laz99] R. Lazić. *A semantic study of data-independence with applications to the mechanical verification of concurrent systems*. PhD thesis, Oxford University, 1999.

[LD00] Gary Leavens and Krishna Dhara. Concepts of behavioral subtyping and a sketch of their extension to Component-Based systems. In *Foundations of Component-Based Systems*, pages 113–135. Cambridge University Press, 2000.

[Lev95] N. Leveson. *Safeware: System Safety and Computers.* Addison-Wesley, 1995.

[LMC10] C. Lambertz and M. E. Majster-Cederbaum. Port protocols for deadlock-freedom of component systems. In S. Bliudze, R. Bruni, D. Grohmann, and A. Silva, editors, *ICE*, volume 38 of *EPTCS*, pages 7–11, 2010.

[LW94] B. H. Liskov and J. M. Wing. A Behavioural Notion of Subtyping. *ACM Transactions on Programming Languages and Systems*, **16**(6), 1994.

[Mah90] M. Mahoney. The roots of software engineering. *CWI Quarterly*, 3(4):325–334, 1990.

[MB05] S. Matougui and A. Beugnard. How to Implement Software Connectors? A Reusable, Abstract and Adaptable Connector. In *IFIP WG 6.1 International Conference in Distributed Applications and Interoperable Systems*, volume 3543 of *Lecture Notes in Computer Science*, pages 83–94. Springer, 2005.

[MCM07] M. Majster-Cederbaum and M. Martens. Robustness in interaction systems. In *27th International Conference on Formal Methods for Networked and Distributed Systems*, volume 4574 of *Lecture Notes of Computer Science*, pages 325–340. Springer, 2007.

[MCM08] M Majster-Cederbaum and M. Martens. Compositional analysis of deadlock-freedom for tree-like component architectures. In *8th ACM international conference on Embedded software*, pages 199–206. ACM, 2008.

[MCMM07] M. Majster-Cederbaum, M. Martens, and C. Minnameier. A polynomial-time checkable sufficient condition for deadlock-freedom of component-based systems. *SOFSEM 2007: Theory and Practice of Computer Science*, pages 888–899, 2007.

[MCMM08] M. Majster-Cederbaum, M. Martens, and C. Minnameier. Liveness in Interaction Systems. *Electronic Notes in Theoretical Computer Science*, 215:57–74, 2008.

[MH05] P. Merson and S. Hissam. Predictability by construction. In *20th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications*, pages 134–135. ACM, 2005.

[Mic11] Microsoft Developer Network. Component object model technologies. Technical report, http://www.microsoft.com/com, 2011.

[Min07] C. Minnameier. Local and global deadlock-detection in component-based systems are NP-hard. *Information Processing Letters*, 103(3):105–111, 2007.

[MJG$^+$10] A. Mota, J. Jesus, A. Gomes, F. Ferri, and E. Watanabe. Evolving a Safe System Design Iteratively. In *29th International Conference Computer Safety, Reliability, and Security*, volume 6351 of *Lecture Notes in Computer Science*, pages 361–374. Springer, 2010.

[MK96] J. Magee and J. Kramer. Dynamic structures in software architecture. In *4th Symposium On the Foundations of Software Engineering*, pages 3–14. ACM Press, 1996.

[Mor94] C. Morgan. *Programming from Specifications*. Prentice-Hall, 1994.

[MT00] N. Medvidovic and R. Taylor. A classification and comparison framework for software architecture description languages. *Transactions on Software Engineering*, 26(1):70–93, 2000.

[MW97] J.M.R. Martin and P.H. Welch. A design strategy for deadlock-free concurrent systems. *Transputer Communications*, 3(4):215–232, 1997.

[Nie93] O. Nierstrasz. Regular types for active objects. *ACM Sigplan Notices*, 28(10):1–15, 1993.

[NSM12] S. Nogueira, A. C. A. Sampaio, and A. C. Mota. Test generation from state based use case models. *Formal Aspects of Computing*, pages 1–50, 2012.

[Obj07] Object Management Group. Unified Modeling Language, Superstructure, V2.1.2. Technical Report formal/2007-11-02, OMG, 2007. OMG Adopted Specification.

[Oli06] M. V. M. Oliveira. *Formal Derivation of State-Rich Reactive Programs using* ***Circus***. PhD thesis, Department of Computer Science, University of York, 2006.

[ORS+12a] M. V. M. Oliveira, R. T. Ramos, A. C. A. Sampaio, A. C. Mota, P. R. G. Antonino, and A. W Roscoe. An Exercise on Strong Output Decisiveness and Input Determinism. Technical report, Centro de Informática - Universidade Federal de Pernambuco, 2012.

[ORS+12b] M. V. M. Oliveira, R. T. Ramos, A. C. A. Sampaio, A. C. Mota, P. R. G. Antonino, and A. W Roscoe. Systematic Development of Constructive Component-based Systems: a Quantitative Analysis. Technical report, 2012. http://www.dimap.ufrn.br/˜marcel/.

[OZC11] M. V. M. Oliveira, F. Zeyda, and A. L. C. Cavalcanti. A Tactic Language for Refinement of State-rich Concurrent Specifications. *Science of Computer Programming*, 76(9):792 – 833, 2011.

[PA98] G. Papadopoulos and F. Arbab. Coordination models and languages. *Advances in Computers - The Engineering of Large Systems*, 46:330–401, 1998.

[Pla05] F. Plasil. Enhancing component specification by behavior description: the SOFA experience. In *4th international symposium on Information and communication technologies*, page 190. Trinity College Dublin, 2005.

[PV02] F. Plasil and S. Visnovsky. Behavior protocols for software components. *IEEE Transactions on Software Engineering*, 28(11):1056–1076, 2002.

[Ram11] Rodrigo Teixeira Ramos. *Systematic Development of Trustworthy Component-based Systems*. PhD thesis, Center of Informatics - Federal University of Pernambuco, Brazil, 2011.

[RM04] R. Roshandel and N. Medvidovic. Multi-View software component modeling for dependability. In *Architecting Dependable Systems II*, volume 3069 of *Lecture Notes in Computer Science*. Springer, 2004.

[Ros98] A. W. Roscoe. *The Theory and Practice of Concurrency*. Prentice-Hall Series in Computer Science. Prentice-Hall, 1998.

[Ros05] A. W. Roscoe. The pursuit of buffer tolerance. Technical report, Oxford University, may 2005.

[Ros10] A.W. Roscoe. *Understanding Concurrent Systems*. Springer-Verlag New York, Inc., New York, NY, USA, 1st edition, 2010.

[RSM08] R. Ramos, A. Sampaio, and A. Mota. Framework composition conformance via refinement checking. In *ACM Symposium on Applied computing*, pages 119–125. ACM, 2008.

[RSM09] R. Ramos, A. Sampaio, and A. Mota. Systematic development of trustworthy component systems. In *2nd World Congress on Formal Methods*, volume 5850 of *Lecture Notes in Computer Science*, pages 140–156. Springer, 2009.

[RSM10] R. Ramos, A. Sampaio, and A. Mota. Conformance notions for the coordination of interaction components. *Science of Computer Programming*, 75(5):350–373, 2010.

[SCHS10] A. Sherif, A. L. C. Cavalcanti, J. He, and A. C. A. Sampaio. A process algebraic framework for specification and validation of real-time systems. *Formal Aspects of Computing*, 22(2):153 – 191, 2010.

[SD01] G. Smith and J. Derrick. Specification, refinement and verification of concurrent systems—an integration of Object-Z and CSP. *Formal Methods in Systems Design*, **18**:249–284, May 2001.

[SGW94] B. Selic, G. Gullekson, and P. Ward. *Real-Time Object-Oriented Modeling*. John Wiley & Sons, Inc., 1994.

[Sif10] J. Sifakis. Component-Based Construction of Heterogeneous Real-Time Systems in Bip. *The Future of Software Engineering*, page 150, 2010.

[Spi92] J. M. Spivey. *The Z Notation: A Reference Manual*. Prentice-Hall, 2nd edition, 1992.

[Spi04] B. Spitznagel. *Compositional Transformation of Software Connectors*. PhD thesis, Carnegie Mellon University, 2004. Number: CMU-CS-04-128.

[SR98] B. Selic and J. Rumbaugh. Using UML for modeling complex RealTime systems. Technical report, Rational Software Corporation, 1998.

[VVR06] A. Vallecillo, V.T. Vasconcelos, and A. Ravara. Typing the behavior of software components using session types. *Fundamenta Informaticae*, 73(4):583–598, 2006.

[Wal03] Kurt C. Wallnau. Volume III: a technology for predictable assembly from certifiable components. Technical Report CMU/SEI-2003-TR-009, Software Engineering Institute, Carnegie Mellon University, 2003.

[WCC$^+$12] Jim Woodcock, Ana Cavalcanti, Joey Coleman, André Didier, Peter Gorm Larsen, Alvaro Miyazawa, and Marcel Oliveira. CML Definition 0. Technical Report D23.1, COMPASS Deliverable, June 2012.

[WCF$^+$12] J. Woodcock, A. Cavalcanti, J. Fitzgerald, P. Larsen, A. Miyazawa, and S. Perry. Features of CML: a Formal Modelling Language for Systems of Systems. In *Proceedings of the 7th International Conference on System of System Engineering*, volume 6 of *IEEE Systems Journal*. IEEE, July 2012.

[WD96] J. C. P. Woodcock and J. Davies. *Using Z—Specification, Refinement, and Proof*. Prentice-Hall, 1996.

[Weh00] H. Wehrheim. Specification of an automatic manufacturing system: A case study in using integrated formal methods. In *3rd Internationsl Conference Fundamental Approaches to Software Engineering*, volume 1783 of *Lecture Notes in Computer Science*, pages 334–348. Springer, 2000.

[Weh03] H. Wehrheim. Behavioral subtyping relations for active objects. *Formal Methods in System Design*, 23(2):143–170, 2003.

[ZKL10] N. Zhan, E. Kang, and Z. Liu. Component publications and compositions. *Unifying Theories of Programming*, pages 238–257, 2010.

[ZM10] H. Zeng and H. Miao. Deadlock Detection for Parallel Composition of Components. *Computer and Information Science*, pages 23–34, 2010.