Local Livelock Analysis of Component-Based Models

M. S. Conserva Filho[1], M. V. M. Oliveira[1], A. Sampaio[2], and Ana Cavalcanti[3]

[1] Universidade Federal do Rio Grande do Norte – Brazil
madiel@ppgsc.ufrn.br, marcel@dimap.ufrn.br
[2] Universidade Federal de Pernambuco – Brazil,
[3] University of York – UK

Abstract. In previous work we have proposed a correct-by-construction approach for building deadlock-free CSP models. It contains a comprehensive set of composition rules that capture safe steps in the development of concurrent systems. In this paper, we extend that work by proposing and implementing a strategy for establishing livelock freedom based on constructive rules similar to those that ensure the absence of deadlock. Our method is based solely on the local analysis of the minimum sequences that lead the CSP model back to its initial state. The effectiveness of our livelock-analysis technique is demonstrated via three case studies. We compare the performance of our approach with that of two other techniques for livelock freedom verification: FDR2 and SLAP.

Keywords: Component-Based Systems, Local Analysis, Livelock.

1 Introduction

Component-based System Development (CBSD) has been used to deal with the increasing complexity of software. It focus on the construction of systems from reusable and independent components [1]. Its correct application, however, relies on the trust in the behaviour of the components and in the emergent behaviour of the composed components because failures may arise if the composition does not preserve essential properties, especially in concurrent systems.

In [9], we have proposed a systematic design of CBSD that integrates components via asynchronous compositions, mediated by buffers, considering a grey-box style of composition [2], in which services that cannot be accessed by other components remain visible to the environment. This strategy is based on safe composition rules that guarantee, by construction, deadlock freedom. The absence of livelock is trivially ensured since the basic components are, by definition, livelock-free, and no operator that may introduce such a behaviour is used. The approach is underpinned by the process algebra CSP [4, 10], a well established formal notation for modelling and verifying concurrent systems. We provided a component model, $\mathcal{BRIC}$, that imposes constraints on the components and their interactions. Each component is represented by a tuple, where one of the elements is the behaviour of the system described as a CSP process.

This paper focuses on livelock analysis for asynchronous CSP models that perform black-box compositions. It defines a component notion that seems better aligned to CBSD, in which the internal services of components are hidden from its environment. This, however, may introduce livelock, a clearly undesirable behaviour. A system is livelock-free if there exists no state from which it may compute through an infinite sequence of internal actions. The traditional livelock analysis performs a global analysis of an internal representation of a model as a labelled transition system, in order to verify that such a state cannot be reached [10]. This strategy is fully automated, for instance, in FDR2 [5]. One alternative is to make a static analysis of the syntactic structure of a system, proposing syntactic rules either to classify CSP systems as livelock-free or to report an inconclusive result. This strategy is implemented in SLAP [7]. Another promising strategy, which is the basis of compositional approaches, performs a local analysis that verifies only some parts of the system. It can identify problems before compositions, predicting, by construction, global properties based on known local properties of the composing components. Locality provides an alternative to circumvent the state explosion generated by the interaction of components and allows us to identify livelock before composition.

In this paper, we present a technique for constructing livelock free systems in $\mathcal{BRIC}$ using local analysis. We consider livelock freedom of $\mathcal{BRIC}$ components in the context of black-boxes rather than grey-boxes compositions adopted in [9]. We introduce side conditions that guarantee, by-construction, that the $\mathcal{BRIC}$ composition rules, which ensure deadlock freedom, also ensure livelock freedom. The verification of these conditions use metadata that allow us to record partial results of verification, decreasing the overall analysis effort. Our strategy supports a systematic development that rules out designs with livelock. We consider two versions of $\mathcal{BRIC}$: $\mathcal{BRIC}^*$, in which asynchronicity is achieved using finite buffers, and $\mathcal{BRIC}^\infty$, which uses infinite buffers. The possibility of introducing livelock is directly related to the finiteness of the buffer. We also present a comparative analysis of the performance of our strategy with respect to those implemented in FDR2 and in SLAP, based on three case studies.

In the next section, we introduce CSP. Section 3 presents the component model $\mathcal{BRIC}$ that defines the building blocks of our systematic development approach. In Section 4, we introduce our approach for livelock-free composition in $\mathcal{BRIC}$ based on local analysis. Its performance is evaluated in Section 5. Finally, we draw our conclusions, and discuss future work in Section 6.

2 CSP

CSP is one of the most important formalisms for modelling and verifying concurrent reactive systems. This process algebra can be used to describe systems as interacting components: independent entities called processes that interact with each other exchanging atomic, instantaneous and synchronous messages, represented by events. The main CSP constructs used in this paper are presented below. Further information can be found in [4, 10].

There are two basic CSP processes: SKIP and STOP. The former represents the terminating process, and the latter deadlocks. The prefixing $a \rightarrow P$ is initially able to perform only the simple event a, and behaves like process P after that. Events may also be compound. For instance, $b.n$ is composed by the channel b and the value n. If we assume that the type of b is the set $\{1, 2\}$, the production $\{\!|\ b\ |\!\}$ returns the set of all events on b, $\{b.1, b.2\}$. Communications may be considered as inputs and outputs: $c!x$ represents an output on some channel c, and $c?x$ is the syntax for an input. The process $g \ \&\ P$ behaves as P if the predicate g is true. Otherwise, it behaves like STOP.

The process $P \ \Box\ Q$ is an external choice between process P and Q: the environment needs to make the choice by communicating an initial event to one of the processes. When the environment has no control over the choice, we have an internal choice $P \sqcap Q$. The process $P;\ Q$ combines the processes P and Q in sequence. The process **if** b **then** P **else** Q behaves as P if b holds and as Q otherwise. The parallel composition $P \parallel_X Q$ synchronises P and Q on the events in the set X; events that are not listed in X occur independently. The interleaving $P \ |||\ Q$ runs the processes independently.

The process $P[[a \leftarrow b]]$ behaves like P except that all occurrences of a in P are replaced by b. The hiding process $P \setminus X$ behaves like P, but all events in the set X are hidden and turned into internal actions, which are not visible to the environment. For example, $P = (a \rightarrow P) \setminus \{a\}$ is a divergent process that indefinitely performs the event a without communicating with its environment.

In order to illustrate some CSP constructs, we use a classical example of a concurrent system, the dining philosophers [10], which is used throughout this paper. It consists of philosophers that try to acquire a pair of shared forks in order to eat. The philosophers are sat at a table and there is a fork between each pair of philosophers. Each philosopher must pick up both forks before eating.

$$
\begin{array}{l}
\mathit{datatype}\ EV = \mathit{up} \mid \mathit{down} \\
\mathit{datatype}\ LF = \mathit{thinks} \mid \mathit{eats} \\
\mathit{channel}\ \mathit{fk}1, \mathit{fk}2, \mathit{pfk}1, \mathit{pfk}2 : EV \\
\mathit{channel}\ \mathit{life} : LF \\
\mathit{Fork} = (\mathit{fk}1.\mathit{up} \rightarrow \mathit{fk}1.\mathit{down} \rightarrow \mathit{Fork}) \ \Box\ (\mathit{fk}2.\mathit{up} \rightarrow \mathit{fk}2.\mathit{down} \rightarrow \mathit{Fork}) \\
\mathit{Phil} = \mathit{life}.\mathit{thinks} \rightarrow \mathit{pfk}1.\mathit{up} \rightarrow \mathit{pfk}2.\mathit{up} \rightarrow \mathit{life}.\mathit{eats} \rightarrow \\
\qquad\quad \mathit{pfk}1.\mathit{down} \rightarrow \mathit{pfk}2.\mathit{down} \rightarrow \mathit{Phil}
\end{array}
$$

The process Fork ensures that two philosophers cannot hold a fork simultaneously. It offers a deterministic choice between the events $\mathit{fk}1.\mathit{up}$ and $\mathit{fk}2.\mathit{up}$, where $\mathit{fk}1$ and $\mathit{fk}2$ are channels of type EV. The process Phil represents the life cycle of a philosopher: before eating, the philosopher thinks and picks the forks up. After eating, the philosopher puts the forks down.

There are three well-established semantic models of CSP: traces ($\mathcal{T}$), stable failures ($\mathcal{F}$), and failures-divergences ($\mathcal{FD}$) [10]. The set $\mathit{traces}(P)$ contains all possible sequences of events in which P can engage. The set $\mathit{failures}(P)$ contains all the failures of P, that is, pairs (s, X), where s is a trace of P and X is a set of events which P may refuse after performing s. The failures-divergences is the most satisfactory model for analysing liveness properties of a CSP process. In $\mathcal{FD}$, a process P is represented by the pair $(\mathit{failures}_{\perp}(P), \mathit{divergences}(P))$. The set $\mathit{failures}_{\perp}(P)$ contains all stable failures of P, and additional failures that record that P can refuse anything after diverging. The set $\mathit{divergences}(P)$ contains all traces of P that lead it to a divergent behaviour and all extensions of those traces. A process P is divergence-free if, and only if, $\mathit{divergences}(P) = \emptyset$.

3 $\mathcal{BRIC}$

The $\mathcal{BRIC}$ component model [9] has been originally proposed to ensure, by construction, the absence of deadlock. It is an algebra that has contracts as operands and composition rules as operators. A component contract, whose definition is presented below, is a tuple and encapsulates a component in $\mathcal{BRIC}$.

Definition 1 (Component contract). *A component contract* $\mathit{Ctr} : \langle \mathcal{B}, \mathcal{R}, \mathcal{I}, \mathcal{C} \rangle$ *comprises its behaviour* $\mathcal{B}$*, which is described as a restricted form of CSP process, I/O process, described below, a set of channels* $\mathcal{C}$*, a set of data types* $\mathcal{I}$*, and a total function* $\mathcal{R} : \mathcal{C} \to \mathcal{I}$ *from channels to their types.*

We use $\mathcal{B}_{Ctr}$, $\mathcal{R}_{Ctr}$, $\mathcal{I}_{Ctr}$ and $\mathcal{C}_{Ctr}$ to denote the elements of the contract Ctr. The behaviour $\mathcal{B}_{Ctr}$ is represented by an I/O process, which is defined as follows, where we use α_P to denote the set of events that P can communicate.

Definition 2 (I/O Process). *We say a CSP process* P *is an I/O process if:*

- *whenever* $c.x \in \alpha_P$*, then* c *is either an input or an output channel;*
- P *has infinite traces (but finite state space);*
- P *is divergence free;*
- P *is input deterministic, that is, after every trace of* P*, if a set of input events of* P *may be offered to the environment, they may not be refused by* P *after the same trace;*
- P *is strongly output decisive, that is, all choices (if any) among output events on a given channel in* P *are internal.*

All channels of an I/O process are either input or output channels. I/O processes are also non-terminating processes but, for practical purposes in model checking, they have finite state spaces, and are divergence free. Input determinism and strong output decisiveness are not relevant in the context of livelock analysis. For this reason, we omit their formal definitions, which can be found in [8].

We illustrate the compositional development of $\mathcal{BRIC}$ with the construction of an asymmetric dining table with 2 philosophers and 2 forks. The behaviour of each philosopher and each fork is represent as a process Phil_i or Fork_i, where $i \in \{1, 2\}$. The channels fk, pfk, both of type $\mathit{ID.ID.EV}$, and lf of type $\mathit{ID.LF}$, where $\mathit{ID} : \{1, 2\}$, distinguish each philosopher and each fork, whose behaviours are described as an instantiation of Phil and Fork described in Section 2.

$$\begin{aligned}
Fork_1 &= Fork\,[[fk1 \leftarrow fk.1.1, fk2 \leftarrow fk.1.2]]\\
Fork_2 &= Fork\,[[fk1 \leftarrow fk.2.2, fk2 \leftarrow fk.2.1]]\\
Phil_1 &= Phil\,[[life \leftarrow lf.1, pfk1 \leftarrow pfk.1.1, pfk2 \leftarrow pfk.2.1]]\\
Phil_2 &= Phil\,[[life \leftarrow lf.2, pfk1 \leftarrow pfk.2.2, pfk2 \leftarrow pfk.1.2]]
\end{aligned}$$

As all forks and philosophers are represented by one process with indices on its channels, there is a separate definition for each component contract. For example, the contracts Ctr_{Fork_1} and Ctr_{Phil_1} are:

$$\begin{aligned}
Ctr_{Fork_1} &= \langle Fork_1, \{fk.1.1 \rightarrow EV, fk.1.2 \rightarrow EV\}, \{EV\}, \{fk.1.1, fk.1.2\}\rangle\\
Ctr_{Phil1} &= \langle Phil_1, \{lf.1 \rightarrow LF, pfk.1.1 \rightarrow EV, pfk.2.1 \rightarrow EV\}, \{LF, EV\},\\
&\qquad \{lf.1, pfk.1.1, pfk.1.2\}\rangle
\end{aligned}$$

The contract Ctr_{Fork_1} has a behaviour defined by $Fork_1$, and two channels: $fk.1.1$ and $fk.1.2$, both of type EV. The behaviour of the contract Ctr_{Phil_1} is $Phil_1$. This contract has three channels, $lf.1$ of type LF, and $pfk.1.1$ and $pfk.2.1$ of type EV.

In $\mathcal{BRIC}$, the asynchronous communications are achieved with the use of (possibly infinite) buffers as intermediate elements of the composition. According to [12], a general buffer in CSP is a process with two channels of the same type: one input and one output. The buffer copies information from its input channel to its output channel, preserving order and without loss. The buffer never refuses inputs when it is empty, considering a finite buffer. It also never refuses outputs when it is non-empty.

Buffers could be considered as regular connectors, but, due to their importance, they are considered as integral part of this strategy. Furthermore, in the original approach, the finiteness of the buffer is irrelevant for deadlock analysis. This is achieved using the notion of *buffer tolerance* [11], which refers to systems into which buffers may be introduced onto their channels without introducing any extra error. Such systems can be analysed in the absence of buffers, considering a synchronous semantics. And, then, these properties are still valid in analogue systems (with buffers introduced onto their channels) with asynchronous semantics. However, for our livelock freedom analysis in $\mathcal{BRIC}$, the finiteness of the buffer is very important, as we discussed in the previous chapter, since the possibility of introducing livelock is directed related to the finiteness of the buffer. Figure 1 illustrates how buffers are represented in this work.

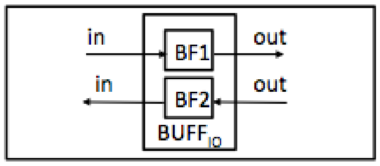

Fig. 1. Buffer Component

In Figure 1, we observe a general component, $BUFF_{IO}$, which includes two internal buffers that do not interact between themselves. They directly communicate with the environment (other components) with no interference from each other. These buffers interact with the environment via two channels, *in* and *out*. We use these internal buffers to perform asynchronous bidirectional communication, which convey information in both directions. As we illustrate in Figure 1, each buffer is used for one direction.

Below, we present the CSP specification of both finite and infinite buffers.

$$\begin{array}{l}
BUFF_{IO}^{\infty} = BF1 \;|||\; BF2 \\
BF1 = \\
\quad \mathsf{let}\ B(\langle\rangle) = in?x \rightarrow B(\langle x \rangle) \\
\qquad\quad B(s) = (out!head(s) \rightarrow B(tail(s))) \\
\qquad\qquad\qquad \Box\ in?x \rightarrow B(s \frown \langle x \rangle)) \\
\quad \mathsf{within}\ B(\langle\rangle) \\
BF2 = \\
\quad \mathsf{let}\ B(\langle\rangle) = out?x \rightarrow B(\langle x \rangle) \\
\qquad\quad B(s) = (in!head(s) \rightarrow B(tail(s))) \\
\qquad\qquad\qquad \Box\ out?x \rightarrow B(s \frown \langle x \rangle)) \\
\quad \mathsf{within}\ B(\langle\rangle)
\end{array}$$

$$\begin{array}{l}
BUFF_{IO}^{*} = BF1 \;|||\; BF2 \\
BF1 = \\
\quad \mathsf{let}\ B(\langle\rangle) = in?x \rightarrow B(\langle x \rangle) \\
\qquad\quad B(s) = (out!head(s) \rightarrow B(tail(s))) \\
\qquad\qquad\qquad \Box\ (\#s < 1\ \&\ in?x \rightarrow B(s \frown \langle x \rangle)) \\
\quad \mathsf{within}\ B(\langle\rangle) \\
BF2 = \\
\quad \mathsf{let}\ B(\langle\rangle) = out?x \rightarrow B(\langle x \rangle) \\
\qquad\quad B(s) = (in!head(s) \rightarrow B(tail(s))) \\
\qquad\qquad\qquad \Box\ (\#s < 1\ \&\ out?x \rightarrow B(s \frown \langle x \rangle)) \\
\quad \mathsf{within}\ B(\langle\rangle)
\end{array}$$

In the specifications above, $BUFF_{IO}^{\infty}$ and $BUFF_{IO}^{*}$ are CSP processes with two internal buffers, $BF1$ and $BF2$, that are put in interleave. The functions $head(s)$ and $tail(s)$ return the head and the tail of a non-empty sequence s, respectively. In order to tune the verification of these study cases, we use the finite buffers of size 1, which does not change the behaviour of studies.

In $\mathcal{BRIC}$, we have two types of component composition: binary composition and unary composition. The former is defined below. It provides an asynchronous interaction on channels *ic* and *oc* between two contracts Ctr_1 and Ctr_2 mediated by a (possibly infinite) bi-directional buffer ($BUFF_{IO}$).

Definition 3 (Asynchronous Binary Composition). *Let Ctr_1 and Ctr_2 be two distinct component contracts with disjoint sets of channels ($\mathcal{C}_{Ctr_1} \cap \mathcal{C}_{Ctr_2} =$*

$\emptyset$), and ic and oc be channels within $\mathcal{C}_{Ctr_1}$ and $\mathcal{C}_{Ctr_2}$, respectively. The asynchronous binary composition of Ctr_1 and Ctr_2 is given by:

$$Ctr_1\,{}_{\langle ic\rangle} \asymp {}_{\langle oc\rangle}\, Ctr_2 = \langle((\mathcal{B}_{Ctr_1} \mid\mid\mid \mathcal{B}_{Ctr_2}) \parallel_{\{|ic,oc|\}} BUFF_{IO}), \mathcal{R}_{Ctr_3}, \mathcal{I}_{Ctr_3}, \mathcal{C}_{Ctr_3}\rangle$$

where $\mathcal{C}_{Ctr_3} = (\mathcal{C}_{Ctr_1} \cup \mathcal{C}_{Ctr_2}) \setminus \{ic, oc\}$, $\mathcal{R}_{Ctr_3} = \mathcal{C}_{Ctr_3} \lhd (\mathcal{R}_{Ctr_1} \cup \mathcal{R}_{Ctr_2})$, and $\mathcal{I}_{Ctr_3} = \operatorname{ran}(\mathcal{R}_{Ctr_3})$.

The behaviour of a binary composition is defined as the synchronisation of the behaviour of Ctr_1 and Ctr_2 via a (possibly infinite) bi-directional buffer. The channels used in the composition are not offered to the environment in further compositions ($\mathcal{C}_{Ctr_3}$). The operator $\lhd$ stands for domain restriction and is used to restrict the mapping from channels to interfaces ($\mathcal{R}_{Ctr_3}$) and, furthermore, to restrict the set of interfaces of the resulting contract ($\mathcal{I}_{Ctr_3}$).

Unary compositions are used to assemble channels of a single component Ctr.

Definition 4 (Asynchronous Unary Composition). *Let Ctr be a component contract, and ic and oc be two distinct channels within $\mathcal{C}_{Ctr}$. The asynchronous unary composition of Ctr is defined as:*

$$Ctr \asymp\Big|_{\langle ic\rangle}^{\langle oc\rangle} = \langle(\mathcal{B}_{Ctr} \parallel_{\{|ic,oc|\}} BUFF_{IO}), \mathcal{R}_{Ctr}, \mathcal{I}_{Ctr}, \mathcal{C}_{Ctr}\rangle$$

where $\mathcal{C}_{Ctr} = (\mathcal{C}_{Ctr} \setminus \{ic, oc\})$, $\mathcal{R}_{Ctr} = \mathcal{C}_{Ctr} \lhd \mathcal{R}_{Ctr}$, and $\mathcal{I}_{Ctr} = \operatorname{ran} \mathcal{R}_{Ctr}$.

The $\mathcal{BRIC}$ composition rules proposed to ensure deadlock freedom by construction are: interleave, communication, feedback and reflexive. The interleave composition aggregates two independent contracts such that, after composition, they do not communicate with each other.

Definition 5 (Interleave composition). *Let Ctr_1 and Ctr_2 be two component contracts, such that $\mathcal{C}_{Ctr_1} \cap \mathcal{C}_{Ctr_2} = \emptyset$. The interleave composition of Ctr_1 and Ctr_2 is given by $Ctr_1 \, [|||] \, Ctr_2 = Ctr_1\,{}_{\langle\rangle} \asymp {}_{\langle\rangle}\, Ctr_2$.*

In this composition, components do not share any channel and no synchronisation is enforced. It is a particular kind of composition that involves no communication. In our example, philosophers and forks can be interleaved separately: $Forks = Ctr_{Fork_1} \, [|||] \, Ctr_{Fork_2}$ and $Phils = Ctr_{Phil_1} \, [|||] \, Ctr_{Phil_2}$. These compositions are valid since the contracts have disjoint channels.

The second rule is based on the traditional way to compose two components, attaching two components connecting two channels, one from each component. Here, Σ is the finite set of all events and $P \upharpoonright X = P \setminus (\Sigma \setminus X)$ restricts the behaviour of P to a set of events X by hiding all events but those in X.

Definition 6 (Communication composition). *Let Ctr_1 and Ctr_2 be two component contracts, and ic and oc two channels, such that $ic \in \mathcal{C}_{Ctr_1}$ and $oc \in \mathcal{C}_{Ctr_2}$, $\mathcal{C}_{Ctr_1} \cap \mathcal{C}_{Ctr_2} = \emptyset$, and $\mathcal{B}_{Ctr_1} \upharpoonright \{ic\}$ and $\mathcal{B}_{Ctr_2} \upharpoonright \{oc\}$ are strong compatible. The communication composition of Ctr_1 and Ctr_2 is defined as:*

$$Ctr_1[ic \leftrightarrow oc]Ctr_2 = Ctr_1\,{}_{\langle ic\rangle} \asymp {}_{\langle oc\rangle}\, Ctr_2.$$

The proviso of strong compatibility ensures that the outputs of each process is always accepted by the other process. Formally, considering that I_P^s and O_P^s denote the inputs and outputs of a process P after a trace s, respectively, P and Q are strong compatible if, and only if:

$$\forall\, s : traces(P) \cap traces(Q) \bullet (O_P^s \neq \emptyset \vee O_Q^s \neq \emptyset) \wedge O_P^s \subseteq I_Q^s \wedge O_Q^s \subseteq I_P^s$$

In our example, we are able to compose the contracts *Forks* and *Phils* using the communication composition: $PComm = Forks[fk.1.1 \leftrightarrow pfk.1.1]Phils$. The resulting contract includes all philosophers and forks. The remaining connections that are needed to complete the dining table require the connection of two channels of the same component. For this reason, $\mathcal{BRIC}$ also provides unary compositions that can be used for such connections and enables the construction of systems with cyclic topologies. Due to the existence of possible cycles, however, new conditions are required to preserve deadlock freedom.

The unary composition rules are feedback and reflexive. The feedback composition represents the simpler unary composition case, where two channels of the same component are assembled, but do not introduce a new cycle [9]. The requirement on the independence of the channels guarantees that no cycles are introduced. A channel c_1 is independent of a channel c_2 in a process when any communication on c_1 does not interfere with the communications on c_2, and vice-versa; hence, both channels are independently offered to the environment.

Definition 7 (Feedback composition). *Let Ctr be a component contract, and ic and oc two communication channels from $\mathcal{C}_{Ctr}$ that are independent in $\mathcal{B}_{Ctr}$, and such that $\mathcal{B}_{Ctr} \upharpoonright ic$ and $\mathcal{B}_{Ctr} \upharpoonright oc$ are strong compatible. The feedback composition of Ctr hooking oc to ic is defined as $Ctr[oc \hookrightarrow ic] = Ctr \asymp|_{\langle oc \rangle}^{\langle ic \rangle}$.*

The contract *PComm* contains all forks and philosophers. The channels *fk*.2.2 and *pfk*.1.2, however, are independent in *PComm* because they occur in the interleaved sub-components *Forks* and *Phils*, respectively. We may, therefore, connect these channels using feedback: $PFeed_1 = PComm[pfk.1.2 \hookrightarrow fk.2.2]$. The channels *fk*.2.1 and *pfk*.2.1 are also independent in $PFeed_1$. Intuitively, their connection do not introduce a cycle; we may, therefore, connect these channels using the feedback composition: $PFeed_2 = PFeed_1[pfk.2.1 \hookrightarrow fk.2.1]$.

The reflexive composition deals with more complex compositions that introduce cycles of dependencies in the topology of the system structure, some of which may be undesirable because they introduce divergence.

Definition 8 (Reflexive composition). *Let Ctr be a component contract, and ic and oc two communication channels from $\mathcal{C}_{Ctr}$ such that $\mathcal{B}_{Ctr} \upharpoonright \{ic, oc\}$ is buffering self-injection compatible. The reflexive composition is defined as $Ctr[ic \overset{\frown}{\hookrightarrow} oc] = Ctr \asymp|_{\langle oc \rangle}^{\langle ic \rangle}$.*

The definition of the reflexive composition is similar to that of the feedback composition. It, however, has a stronger proviso that requires buffering self-injection compatibility, which allows one to assembly two dependent channels of a process via a buffer, without introducing deadlocks. This property is similar to the notion of strong compatibility, except for the fact that two distinct channels of the same process must be compatible. Its formalisation can be found in [8].

In our example, we conclude the design of our system using the reflexive composition to connect channels $fk.1.2$ and $pfk.2.2$.

$$PSystem = PFeed_2[fk.1.2 \hookrightarrow pfk.2.2].$$

This connection could not be achieved using feedback because the two channels are not independent in $PFeed_2$. Intuitively, their connection introduces a cycle that causes the dependence between these channels.

4 Livelock Analysis for $\mathcal{BRIC}$

In the $\mathcal{BRIC}$ approach livelock is not an issue because the rules do not hide the composed channels in the CSP behaviour of the resulting contract; they are just removed from the communication channel set, preventing further compositions on them. This gives us a grey-box style of abstraction [2]. We extend the possibilities of performing compositions in $\mathcal{BRIC}$, providing a constructive strategy to perform black-box compositions [13], where the components encapsulate functionality, increasing the abstraction level of the system.

In [9], the concept of livelock is not defined at the component contract level. We define the notion of livelock-free component contract that considers $\mathcal{BRIC}$ components as black-boxes. For that, we consider the component behaviour and the communication channels that are in the component set of visible channels, which are eligible for future compositions. As a result, a component contract Ctr is livelock-free if the CSP process resulting from hiding all channels that are not in the set $\mathcal{C}_{Ctr}$ in the behaviour $\mathcal{B}_{Ctr}$ is divergence free.

Definition 9 (Livelock-free Component Contract). *A component contract $Ctr = \langle \mathcal{B}, \mathcal{R}, \mathcal{I}, \mathcal{C} \rangle$ is livelock-free if, and only if, $divergences(\mathcal{B}_{Ctr} \upharpoonright \mathcal{C}_{Ctr}) = \emptyset$.*

In what follows, we present the definitions used in our livelock analysis technique and describe the local conditions that guarantee livelock-free $\mathcal{BRIC}$ compositions at the component contract level. We make a clear distinction of asynchronous compositions via finite and infinite buffers because the finiteness of the buffer is relevant for detecting the possibility of livelock in asynchronous systems. We consider $\mathcal{BRIC}^*$, which achieves asynchronous compositions via finite buffers, and $\mathcal{BRIC}^\infty$, in which asynchronicity is achieved using infinite buffers.

4.1 Basic Definitions

A livelock-free contract never performs an infinite sequence of internal events without communicating with its environment. Hence, reasoning about divergences requires reasoning about infinite behaviours. Therefore, the first step of our approach identifies the infinite behaviours of a given component. The function $IP(P)$ returns the traces that lead a given process P to a recursion.

Definition 10 (Interaction Patterns). *Let P be a CSP process. The set of interaction patterns is defined as:* $IP(P) = \{t : traces(P) \mid P \equiv_{\mathcal{FD}} (P/t)\}$.

The process P/t (pronounced P after t) represents the behaviour of P after the trace t is performed. The set $IP(P)$ contains all traces of P after which the process (P/t) has the same failures and divergences of P: they are equivalent in the failures-divergences model. Hence, $IP(P)$ gives an infinite set of traces that leads the process P back to its initial state. In our example, the set of interaction patterns of $IP(Fork_1)$ contains the traces that lead this fork to a recursion:

$$\{\langle fk.1.1.up, fk.1.1.down\rangle, \langle fk.1.2.up, fk.1.2.down\rangle,\\ \langle fk.1.1.up, fk.1.1.down, fk.1.1.up, fk.1.1.down\rangle, \ldots\}$$

This set is infinite. Our strategy, however, only needs the set of minimal interaction patterns, which only contains the traces that lead the process to its first recursion. In what follows, we use the function S°, which, given a set of traces S (in our case, interaction patterns), returns the concatenation closure on S, i.e., the set of all sequences we can obtain by taking any subset of traces from the original S and concatenating them together (possibly with repetitions).

$$S^{\circ} = \{t : \Sigma^* \mid (\exists\, ss : \mathrm{seq}(\Sigma^*) \bullet \mathrm{ran}(ss) \subseteq S \wedge t = {}^\frown/\, ss)\}$$

Here, Σ^* is the set of finite sequences of elements of Σ, $\mathrm{seq}(\Sigma^*)$ is the set of finite sequences over Σ^*, ${}^\frown/\, ss$ is the distributed concatenation of all the elements of the sequence of sequences ss, and ran(ss) is the set of the elements of ss.

The set of Minimal Interaction Patterns of a process P, $MIP(P)$, is the minimal set from which we are able to generate the same traces that can be generated from $IP(P)$. Formally, it is a subset of any other subset of interaction patterns S of $IP(P)$, such that $S^{\circ} = IP(P)$.

Definition 11 (Minimal Interaction Patterns). *Let P be a CSP process. The set of minimal interaction patterns of P, $MIP(P)$, is a set such that*

$$(MIP(P))^{\circ} = IP(P) \text{ and } \forall\, S : \mathbb{P}(\Sigma^*) \mid S^{\circ} = IP(P) \bullet MIP(P) \subseteq S.$$

The set of $MIP(P)$ contains the shortest finite traces that lead the process P to its first recursion. In this manner, the process P repeatedly offers these sequences of events. This means that the infinite traces of P are generated only by the infinite concatenation of the traces of $MIP(P)$. In other words, the sequences in $MIP(P)$ provide enough information for deducing the set of infinite traces of P. The formalisation of this semantic property is presented as follows.

$$\forall\, u : \Sigma^{\omega} \bullet (\forall\, s < u \bullet s \in traces(P)) \Rightarrow (u \in MIP(P)^{\omega})\\ \text{where } S^{\omega} = \{t : \Sigma^{\omega} \mid \forall\, s < t \bullet (\exists\, ss : S^* \bullet s < ss)\}$$

The following constructive proposition is based on the calculation of *traces* proposed by Roscoe [10]. It calculates the *MIP* for CSP processes that describe the behaviour of the basic components, which are strictly sequential (possibly with choices) with no hiding. Parallelism is achieved by composing component contracts using the composition rules. We also consider only tail recursion (and no mutual recursion), in which recursive calls may only happen after at least one visible event (guarded tail recursions). In what follows, we use N to denote the process name and $\overline{P}$ to represent the CSP process expression that defines its behaviour. We also use W_1 and W_2 to denote CSP behaviours.

Proposition 1 (Minimal Interaction Patterns Calculation). *Let N be a process name, and $\overline{P}$ its behaviour. Then $MIP(N)$ is given by $MIP_N(\overline{P})$:*

$$
\begin{array}{l}
MIP_N(N) = \{\langle\rangle\} \\
MIP_N(SKIP) = MIP_N(STOP) = \{\} \\
MIP_N(c \to W_1) = \{t : MIP_N(W_1);\; e : \{\!|\; c \;|\!\} \bullet \langle e\rangle \frown t\} \\
MIP_N(W_1 \Box W_2) = MIP_N(W_1 \sqcap W_2) = MIP_N(W_1) \cup MIP_N(W_2) \\
MIP_N(W_1[[R]]) = \bigcup\{t : MIP_N(W_1) \bullet ren(t, R)\} \\
MIP_N(W_1;\; W_2) = \left\{ \begin{array}{r} t_1 : traces(W_1);\; t_2 : MIP_N(W_2) \mid last(t_1) = \checkmark \\ \bullet\, front(t_1) \frown t_2 \end{array} \right\} \\
MIP_N(g \;\&\; W_1) = MIP_N(W_1) \\
MIP_N(\mathbf{if}\;\; g\; \mathbf{then}\;\; W_1 \;\mathbf{else}\;\; W_2) = MIP_N(W_1) \cup MIP_N(W_2)
\end{array}
$$

The sequence $front(t)$ contains all elements of the sequence t but the last one, $last(t)$ returns the last element of t, and the function $ren(t, R)$, presented below, applies the renaming relation on events R to the trace t. For functional renaming, this function returns a singleton set that contains a trace that corresponds to t but replaces every element in the domain of the renaming function by its image. However, relational renaming needs special care because it may turn simple prefixing into an external choice. By way of illustration, for $P = a \to P$, $P[[a \leftarrow b, a \leftarrow c]] = a \to P \Box c \to P$. For this reason, the function *rename* presented below returns a set of traces and we need a distributed union ($\bigcup$) in the definition of MIP_N for renaming (see Proposition 1).

$$
\begin{array}{l}
ren(\langle\rangle, R) = \{\langle\rangle\} \\
ren(\langle e\rangle \frown t, R) = \mathbf{if}\; e \in \mathrm{dom}(R)\; \mathbf{then}\;\; \{e' : R[\{c\}];\; s : ren(t, R) \bullet \langle e'\rangle \frown s\} \\
\qquad\qquad\qquad\qquad \mathbf{else}\; \{s : ren(t, R) \bullet \langle e\rangle \frown s\}
\end{array}
$$

In Proposition 1, when MIP_N is applied to N itself, the result is the empty sequence. With our assumption that the process is guarded tail recursive, this ensures that at this stage a minimum path is recorded. *SKIP* and *STOP* do not contain any *MIP* because they terminate (either successfully or not). The MIP_N of the prefix process $c \to W_1$ is formed by concatenating the sequence $\langle c\rangle$ to the front of the sequences of $MIP_N(W_1)$. The *MIP* of internal and external choices are the union of the MIP_N of the two operands. The *MIP* of $W_1[[R]]$ are those of W_1 replacing all occurrences of the events e in the domain of the renaming relation R by the relational image of $\{e\}$ in R. The $MIP_N(W_1;\; W_2)$ are the ones of W_2 prefixed by the traces of W_1 that lead to termination, but removing $\checkmark$. The calculation of the MIP_N of guarded processes $g \;\&\; W_1$ (and alternation **if** g **then** W_1 **else** W_2) simply ignores the guard g and takes $MIP_N(W_1)$ (and $MIP_N(W_2)$) as the result. As a consequence, our approach may find false negatives because we consider interaction patterns which may not be feasible depending on the evaluation of g. For instance, if we consider a process $P = g \;\&\; a \rightarrow P$, our approach indicates the possibility of divergence in $P \setminus \{a\}$ because we do not analyse the value of g, which determines the existence of either a divergence or a deadlock.

In our example, the calculation of the minimum interaction patterns for $Fork_1$ and $Phil_1$ yields the following result.

$$\begin{aligned} MIP(Fork_1) &= \{\langle fk.1.1.up, fk.1.1.down\rangle, \langle fk.1.2.up, fk.1.2.down\rangle\} \\ MIP(Phil_1) &= \{\langle lf.1.thinks, pfk.1.1.up, pfk.2.1.up, lf.1.eats, \\ &\qquad pfk.1.1.down, pfk.2.1.down\rangle\} \end{aligned}$$

We are now able to infer which channels can be used to compose a livelock-free contract in $\mathcal{BRIC}$. The function *Allowed* identifies all communication channels that can be individually hidden with no introduction of contract livelock.

Definition 12 (*Allowed*)**.** *Let Ctr be a component contract. The set of communication channels of $\mathcal{C}_{Ctr}$ that can be individually hidden with no introduction of divergence is given by Allowed(Ctr) defined below:*

$$\begin{aligned} &Allowed(Ctr) = \\ &\qquad \mathcal{C}_{Ctr} \setminus \{c : \mathcal{C}_{Ctr} \mid \exists\, s : MIP(\mathcal{B}_{Ctr}) \bullet \operatorname{ran}(s) \cap evs(\mathcal{C}_{Ctr}) \subseteq evs(\{c\})\} \end{aligned}$$

The set $evs(cs) = \bigcup\{c : cs \bullet \{\!|\; c \;|\!\}\}$ contains all events produced by the channels in the set cs given as argument.

The set of *Allowed* channels of a given contract *Ctr* contains all communication channels c, such that there is no $MIP(\mathcal{B}_{Ctr})$ composed only by events on c. Using these channels on compositions does not introduce a contract livelock because even after individually hiding the communication on these channels, every member of $MIP(\mathcal{B}_{Ctr})$ still has at least one further external communication on a different channel with the environment. In our example, the sets of allowed channels are $Allowed(Ctr_{Phil_1}) = \{lf.1, pfk.1.1, pfk.2.1\}$ and $Allowed(Ctr_{Fork_1}) = \emptyset$. The latter is empty because every member of $MIP(Fork_1)$ either contains only interactions on $fk.1.1$ or only interactions on $fk.1.2$.

4.2 Conditions for Livelock Freedom in $\mathcal{BRIC}^*$

An interleave composition always results in a livelock-free contract, since the behaviour of both composing contracts are livelock-free by definition, and no communication channel is used in this composition. Therefore, we establish the following theorem that ensures a livelock-free component contract as result of an interleave composition.

Theorem 1 (Livelock free Interleave Composition).
Let Ctr_1 and Ctr_2 be two livelock-free component contracts, then $Ctr_1 \; [|||] \; Ctr_2$ is a livelock-free component contract.

Proof

$divergences(\mathcal{B}_{Ctr_1\,[|||]\,Ctr_2} \upharpoonright \mathcal{C}_{Ctr_1\,[|||]\,Ctr_2}) = \emptyset$

$divergences(\mathcal{B}_{Ctr_1\,[|||]\,Ctr_2} \upharpoonright \mathcal{C}_{Ctr_1\,[|||]\,Ctr_2})$

$=$ [Definition 5]

$divergences(\mathcal{B}_{Ctr_1\,\langle\rangle \asymp \langle\rangle\,Ctr_2} \upharpoonright \mathcal{C}_{Ctr_1\,\langle\rangle \asymp \langle\rangle\,Ctr_2})$

$=$ [Lemma 1]

$divergences(\mathcal{B}_{Ctr_1} \setminus (\Sigma \setminus (\mathcal{C}_{Ctr_1} \cup \mathcal{C}_{Ctr_2})) \,|||\, \mathcal{B}_{Ctr_2} \setminus (\Sigma \setminus (\mathcal{C}_{Ctr_1} \cup \mathcal{C}_{Ctr_2})))$

$=$ [Definition 16 ($divergences(P \parallel_{\{\}} Q)$)]

$\{u \frown v \mid \exists\, s \in traces(\mathcal{B}_{Ctr_1} \setminus (\Sigma \setminus \mathcal{C}_{Ctr_1} \cup \mathcal{C}_{Ctr_2})),$
$\quad t \in traces(\mathcal{B}_{Ctr_2} \setminus (\Sigma \setminus \mathcal{C}_{Ctr_1} \cup \mathcal{C}_{Ctr_2}))$
$\quad \bullet\ u \in (s \parallel_{\{\}} t) \cap\ \Sigma^* \wedge (s \in divergences(\mathcal{B}_{Ctr_1} \setminus (\Sigma \setminus \mathcal{C}_{Ctr_1} \cup \mathcal{C}_{Ctr_2}))$
$\qquad \vee\ t \in divergences(\mathcal{B}_{Ctr_2} \setminus (\Sigma \setminus \mathcal{C}_{Ctr_1} \cup \mathcal{C}_{Ctr_2}))\}$

$=$ [Lemma 2, Ctr_1 is a livelock-free contract and $\Sigma \setminus (\mathcal{C}_{Ctr_1} \cup \mathcal{C}_{Ctr_2}) \subseteq \Sigma \setminus \mathcal{C}_{Ctr_1}$]

$\{u \frown v \mid \exists\, s \in traces(\mathcal{B}_{Ctr_1} \setminus (\Sigma \setminus \mathcal{C}_{Ctr_1} \cup \mathcal{C}_{Ctr_2})),$
$\quad t \in traces(\mathcal{B}_{Ctr_2} \setminus (\Sigma \setminus \mathcal{C}_{Ctr_1} \cup \mathcal{C}_{Ctr_2}))$
$\quad \bullet\ u \in (s \parallel_{\{\}} t) \cap\ \Sigma^* \wedge$
$\qquad (s \in \emptyset \vee t \in divergences(\mathcal{B}_{Ctr_2} \setminus (\Sigma \setminus \mathcal{C}_{Ctr_2} \cup \mathcal{C}_{Ctr_1}))\}$

$=$ [Lemma 2, Ctr_2 is a livelock-free contract and $\Sigma \setminus (\mathcal{C}_{Ctr_2} \cup \mathcal{C}_{Ctr_1}) \subseteq \Sigma \setminus \mathcal{C}_{Ctr_2}$]

$\{u \frown v \mid \exists\, s \in traces(\mathcal{B}_{Ctr_1} \setminus (\Sigma \setminus \mathcal{C}_{Ctr_1} \cup \mathcal{C}_{Ctr_2})),$
$\quad t \in traces(\mathcal{B}_{Ctr_2} \setminus (\Sigma \setminus \mathcal{C}_{Ctr_1} \cup \mathcal{C}_{Ctr_2}))$
$\quad \bullet\ u \in (s \parallel_{\{\}} t) \cap\ \Sigma^* \wedge (s \in \emptyset \vee t \in \emptyset)\}$

$=$ [Property of Sets and Predicate Calculus]

$\{u \frown v \mid false\}$

$=$ [Property of Sets]

$\emptyset$

□

Theorem 1 ensures that the contract resulting from an interleave composition is a livelock-free component contract. This proof is also valid for compositions in $\mathcal{BRIC}^{\infty}$, since no property on the buffer is used in the composition, and no proviso is required from the buffer. The remaining rules, however, only ensure a livelock-free contract under specific conditions.

In the communication composition via finite buffers, $Ctr_1[ic \leftrightarrow oc]^* Ctr_2$, a contract livelock may be introduced because we hide the channels ic and oc used in the composition, since they are removed from the set $\mathcal{C}$ of the resulting component. There are, however, conditions under which this composition is safe.

For instance, we consider the composition $Ctr_{Fork_1}[fk.1.1 \leftrightarrow pfk.1.1]^* Ctr_{Phil_1}$ previously presented. Since the communication is asynchronous, after sending the events $fk.1.1.up$ and $fk.1.1.down$ to the buffer, $Fork_1$ recurses and may send such events to the buffer again before the first ones have been consumed by $Phils_1$ via $pfk.1.1.up$ and $pfk.1.1.down$. This, however, may be done only a finite number of times because the buffer is finite and, at some point, the communications on $pfk.1.1.up$ and $pfk.1.1.down$ will be enforced causing the occurrences, for instance, of the visible events $lf.1.thinks$ and $lf.1.eats$. This composition is, therefore, livelock-free. Along with the finiteness of the buffer, the fact that one of the connecting channels is in the corresponding set of allowed channels ($pfk.1.1 \in Allowed(Ctr_{Phils_1})$) guarantees a resulting livelock-free contract.

We establish below a condition that ensures that a contract livelock is not introduced in a communication composition in $\mathcal{BRIC}^*$.

Theorem 2 (*Livelock-free Finite Communication Compositions*). *Let Ctr_1 and Ctr_2 be two livelock-free component contracts, and ic and oc two channels in $\mathcal{C}_{Ctr_1}$ and $\mathcal{C}_{Ctr_2}$, respectively. The composition $Ctr_1[ic \leftrightarrow oc]^* Ctr_2$ is livelock-free if $ic \in Allowed(Ctr_1)$* ***or*** *$oc \in Allowed(Ctr_2)$.*

Proof

$$\begin{pmatrix} (ic \in Allowed(Ctr_1)) \lor (oc \in Allowed(Ctr_2)) \\ \Rightarrow divergences(\mathcal{B}_{Ctr_1[ic \leftrightarrow oc]Ctr_2} \upharpoonright \mathcal{C}_{Ctr_1[ic \leftrightarrow oc]Ctr_2}) = \emptyset \end{pmatrix}$$

$\equiv$ [Predicate Calculus]

$$\begin{pmatrix} (ic \in Allowed(Ctr_1)) \\ \Rightarrow divergences(\mathcal{B}_{Ctr_1[ic \leftrightarrow oc]Ctr_2} \upharpoonright \mathcal{C}_{Ctr_1[ic \leftrightarrow oc]Ctr_2}) = \emptyset \end{pmatrix}$$

$\land$

$$\begin{pmatrix} (oc \in Allowed(Ctr_2)) \\ \Rightarrow divergences(\mathcal{B}_{Ctr_1[ic \leftrightarrow oc]Ctr_2} \upharpoonright \mathcal{C}_{Ctr_1[ic \leftrightarrow oc]Ctr_2}) = \emptyset \end{pmatrix}$$

$\equiv$ [Lemma 18]

$$true \land \begin{pmatrix} (oc \in Allowed(Ctr_2)) \\ \Rightarrow divergences(\mathcal{B}_{Ctr_1[ic \leftrightarrow oc]Ctr_2} \upharpoonright \mathcal{C}_{Ctr_1[ic \leftrightarrow oc]Ctr_2}) = \emptyset \end{pmatrix}$$

$\equiv$ [Lemma 18]

$true \land true$

$\equiv$ [Predicate Calculus]

$true$

□

Theorem 2 ensures that the contract resulting from a communication composition is a livelock-free component contract.

Regarding unary compositions, due to the finiteness of the buffer, we also only need to check if at least one of the communication channels used in the composition belongs to the set of *Allowed* channels of the contract.

Theorem 3 (***Livelock-free Finite Unary Compositions***)**.** *Let Ctr be a livelock-free component contract, and ic and oc two channels in $\mathcal{C}_{Ctr}$. The compositions $Ctr[ic \hookrightarrow oc]^*$ and $Ctr[ic \overset{-}{\hookrightarrow} oc]^*$ are livelock-free if $ic \in Allowed(Ctr)$* ***or*** *$oc \in Allowed(Ctr)$.*

Proof
As the condition for ensuring a livelock-free contract in feedback compositions and reflexive compositions is the same, we only present the proof for a feedback composition in $\mathcal{BRIC}^*$.

$$\begin{pmatrix} (ic \in Allowed(Ctr)) \vee (oc \in Allowed(Ctr)) \\ \Rightarrow divergences(\mathcal{B}_{Ctr[oc \hookrightarrow ic]} \upharpoonright \mathcal{C}_{Ctr[oc \hookrightarrow ic]}) = \emptyset \end{pmatrix}$$

$\equiv$ [Predicate Calculus]

$$\begin{pmatrix} (ic \in Allowed(Ctr)) \\ \Rightarrow divergences(\mathcal{B}_{Ctr[oc \hookrightarrow ic]} \upharpoonright \mathcal{C}_{Ctr[oc \hookrightarrow ic]}) = \emptyset \end{pmatrix}$$

$\wedge$

$$\begin{pmatrix} (oc \in Allowed(Ctr)) \\ \Rightarrow divergences(\mathcal{B}_{Ctr[oc \hookrightarrow ic]} \upharpoonright \mathcal{C}_{Ctr[oc \hookrightarrow ic]}) = \emptyset \end{pmatrix}$$

$\equiv$ [Lemma 20]

$$true \wedge \begin{pmatrix} (oc \in Allowed(Ctr)) \\ \Rightarrow divergences(\mathcal{B}_{Ctr[oc \hookrightarrow ic]} \upharpoonright \mathcal{C}_{Ctr[oc \hookrightarrow ic]}) = \emptyset \end{pmatrix}$$

$\equiv$ [Lemma 20]

$true \wedge true$

$\equiv$ [Predicate Calculus]

$true$

□

Theorem 3 establishes the conditions under which feedback and reflexive finite compositions are livelock-free.

We now turn our attention to the cases in which neither of the connecting channels are in the set of *Allowed*. For example, let us consider three simple livelock-free contracts Ctr_1, Ctr_2 and Ctr_3 defined as follows.

$$\begin{array}{l} Ctr_1 : \langle \mathcal{B}_{Ctr_1}, \{a \to \mathbb{N}\}, \{\mathbb{N}\}, \{a\} \rangle, where\, \mathcal{B}_{Ctr_1} = a.1 \to a.2 \to \mathcal{B}_{Ctr_1} \\ Ctr_2 : \langle \mathcal{B}_{Ctr_2}, \{b \to \mathbb{N}\}, \{\mathbb{N}\}, \{b\} \rangle, where\, \mathcal{B}_{Ctr_2} = b.2 \to b.3 \to \mathcal{B}_{Ctr_2} \\ Ctr_3 : \langle \mathcal{B}_{Ctr_3}, \{c \to \mathbb{N}\}, \{\mathbb{N}\}, \{c\} \rangle, where\, \mathcal{B}_{Ctr_3} = c.1 \to c.2 \to \mathcal{B}_{Ctr_3} \end{array}$$

The composition $Ctr_1[a \leftrightarrow c]^* Ctr_3$ is valid in $\mathcal{BRIC}$ because a and c are strong compatible. However, neither a or c are allowed in the corresponding contracts; this composition yields a divergent contract. In general, however, this would not necessarily happen. For example, $Ctr_1[a \leftrightarrow b]^* Ctr_2$ would not introduce a contract livelock because the channels would not be able to synchronise. The $\mathcal{BRIC}$ rules, however, require the connecting channels to be strong compatible, that is, at every state of a in $\mathcal{B}_{Ctr_1}$ if $a.n$ is offered, then $b.n$ is also offered by

$\mathcal{B}_{Ctr_2}$. In $Ctr_1[a \leftrightarrow b]Ctr_2$, a and b are not strong compatible. As a consequence of the strong compatibility requirement, there is no case in which neither of the connecting channels are in *Allowed* of their contracts and the $\mathcal{BRIC}$ compositions result in a livelock-free component contract.

In $\mathcal{BRIC}^{\infty}$, the assumption that communications with the buffer will halt at some point because the buffer is full is no longer valid because the buffers are infinite. We, therefore, need stronger conditions to ensure livelock freedom.

4.3 Conditions for Livelock Freedom in $\mathcal{BRIC}^{\infty}$

In the presence of infinite buffers, the conditions for safe compositions are necessarily stronger because one of the contracts may indefinitely interact with the buffer via the connecting channel. For example, let us revisit the example of Section 4.2 replacing the buffer by an infinite one. The communication composition $Ctr_{Fork_1}[fk.1.1 \leftrightarrow pfk.1.1]^{\infty}Ctr_{Phil_1}$ remains asynchronous. After sending $fk.1.1.up$ and $fk.1.1.down$ to the buffer, $Fork_1$ still recurses and may send such events to the buffer again before the first ones has been consumed by $Phils_1$ via $pfk.1.1.up$ and $pfk.1.1.down$. This, however, may now be done indefinitely because the buffer is infinite; there is no guarantee that $Phils_1$ ever consumes any message on $pfk.1.1.up$ and $pfk.1.1.down$ causing the occurrence, for instance, of the visible events $lf.1.thinks$ and $lf.1.eats$. For this reason, the divergence of $Fork_1$ affects the overall composition. Therefore, we need a stronger requirement to ensure contract livelock freedom in a communication composition in $\mathcal{BRIC}^{\infty}$.

Theorem 4 (*Livelock-free Infinite Communication Compositions*). *Let Ctr_1 and Ctr_2 be two livelock-free contracts, and ic and oc two channels in $\mathcal{C}_{Ctr_1}$ and $\mathcal{C}_{Ctr_2}$, respectively. The composition $Ctr_1[ic \leftrightarrow oc]^{\infty}Ctr_2$ is livelock-free if $ic \in Allowed(Ctr_1)$* ***and*** *$oc \in Allowed(Ctr_2)$.*

Proof

$$\left(\begin{array}{l}(ic \in Allowed(Ctr_1)) \wedge (oc \in Allowed(Ctr_2))\\ \Rightarrow divergences(\mathcal{B}_{Ctr_1[ic \leftrightarrow oc]Ctr_2} \upharpoonright \mathcal{C}_{Ctr_1[ic \leftrightarrow oc]Ctr_2}) = \emptyset\end{array}\right)$$

We assume that $ic \in Allowed(Ctr_1) \wedge oc \in Allowed(Ctr_2)$ is true, and prove that:

$$divergences(\mathcal{B}_{Ctr_1[ic \leftrightarrow oc]Ctr_2} \upharpoonright \mathcal{C}_{Ctr_1[ic \leftrightarrow oc]Ctr_2}) = \emptyset$$

$divergences(\mathcal{B}_{Ctr_1[ic \leftrightarrow oc]Ctr_2} \upharpoonright \mathcal{C}_{Ctr_1[ic \leftrightarrow oc]Ctr_2})$

$=$ [Definition 6]

$divergences(\mathcal{B}_{Ctr_1\,\langle ic\rangle \asymp \langle oc\rangle\, Ctr_2} \upharpoonright \mathcal{C}_{Ctr_1\,\langle ic\rangle \asymp \langle oc\rangle\, Ctr_2})$

$=$ [Lemma 3]

$divergences(((\mathcal{B}_{Ctr_1} \interleave \mathcal{B}_{Ctr_2}) \parallel_{\{ic,oc\}} BUFF_{IO}^{\infty}) \setminus HC_{Bin})$

$=$ [Lemma 6, Lemma 4 and Lemma 5]

$$\{(u \setminus HC_{Bin}) \frown t \mid u \in \Sigma^{\omega} \wedge \neg\, (u \setminus HC_{Bin}) : \Sigma^{\omega} \\ \wedge \forall\, s < u \bullet s \in traces((\mathcal{B}_{Ctr_1} \interleave \mathcal{B}_{Ctr_2}) \parallel_{\{ic,oc\}} BUFF^{\infty}_{IO})\}$$

$=$ [Lemma 7 and Definition of $tr^{\infty}(u, (\mathcal{B}_{Ctr_1} \interleave \mathcal{B}_{Ctr_2}) \parallel_{\{ic,oc\}} BUFF^{\infty}_{IO})$ (18)]

$$\{(u \setminus HC_{Bin}) \frown t \mid u \in \Sigma^{\omega} \wedge \neg\, (u \setminus HC_{Bin}) : \Sigma^{\omega} \\ \wedge \forall\, s < u \bullet s \in traces((\mathcal{B}_{Ctr_1} \interleave \mathcal{B}_{Ctr_2}) \parallel_{\{ic,oc\}} BUFF^{\infty}_{IO}) \\ \wedge\, u \setminus HC_{Bin} \;: \Sigma^{\omega}\}$$

$=$ [Property of Sets and Predicate Calculus]

$$\{(u \setminus HC_{Bin}) \frown t \mid false\}$$

$=$ [Property of Sets]

$$\emptyset$$

$\Box$

Theorem 4 ensures that the contract resulting from a communication composition in $\mathcal{BRIC}^{\infty}$ is a livelock-free component contract.

Regarding the unary compositions in $\mathcal{BRIC}^{\infty}$, we have to ensure that the pair of connecting channels can be hidden together. We define the function $Allowed_{Bin}(Ctr)$, which is similar to $Allowed(Ctr)$, but characterises all pairs of channels that can be hidden together without generating a contract livelock.

Definition 13 **($Allowed_{Bin}$).** *Let Ctr be a livelock-free contract. The set of pairs of channels of $\mathcal{C}_{Ctr}$ that can be hidden with no introduction of divergence is given by $Allowed_{Bin}(Ctr)$ defined as:*

$$Allowed_{Bin}(Ctr) = \\ \{c_1, c_2 : \mathcal{C}_{Ctr} \mid \neg\, (\exists\, s : MIP(\mathcal{B}_{Ctr}) \bullet \operatorname{ran}(s) \cap evs(\mathcal{C}_{Ctr}) \subseteq \{\!| \; c_1, c_2 \; |\!\})\}$$

For the same reason, the infiniteness of the buffers, unary compositions in $\mathcal{BRIC}^{\infty}$ have a stronger condition for ensuring livelock freedom. We require both connecting channels to be allowed to be hidden together.

Theorem 5 **(*Livelock-free Infinite Unary Compositions*).** *Let Ctr be a livelock-free contract, and ic and oc two channels in $\mathcal{C}_{Ctr}$. The compositions $Ctr[ic \hookrightarrow oc]^{\infty}$ and $Ctr[ic \,\overset{\curvearrowleft}{\hookrightarrow}\, oc]^{\infty}$ are livelock-free if $(ic, oc) \in Allowed_{Bin}(Ctr)$.*

Proof

As the condition for ensuring a livelock-free contract in feedback compositions and reflexive compositions is the same, we only present the proof for a feedback composition in $\mathcal{BRIC}^{\infty}$.

$$\left(\begin{array}{l} (ic, oc) \in Allowed_{Pair}(Ctr) \\ \Rightarrow divergences(\mathcal{B}_{Ctr[oc \hookrightarrow ic]^{\infty}} \upharpoonright \mathcal{C}_{Ctr[oc \hookrightarrow ic]^{\infty}}) = \emptyset \end{array}\right)$$

We assume that $(ic, oc) \in Allowed_{Pair}(Ctr)$ is true, and prove that:

$divergences(\mathcal{B}_{Ctr[oc \hookrightarrow ic]^\infty} \upharpoonright \mathcal{C}_{Ctr[oc \hookrightarrow ic]^\infty}) = \emptyset$

$divergences(\mathcal{B}_{Ctr[oc \hookrightarrow ic]^\infty} \upharpoonright \mathcal{C}_{Ctr[oc \hookrightarrow ic]^\infty})$
$=$ [Definition 7]
$divergences(\mathcal{B}_{Ctr \prec\!\!\dashv^{\langle ic \rangle}_{\langle oc \rangle}} \upharpoonright \mathcal{C}_{Ctr \prec\!\!\dashv^{\langle ic \rangle}_{\langle oc \rangle}})$
$=$ [Lemma 15]
$divergences((\mathcal{B}_{Ctr} \parallel_{\{ic,oc\}} BUFF^\infty_{IO}) \setminus HC_{Unary})$
$=$ [Lemma 4 and Lemma 16]
$\{(u \setminus HC_{Unary}) ^\frown t \mid u \in \Sigma^\omega \wedge \neg\, (u \setminus HC_{Unary}) : \Sigma^\omega$
$\wedge\, \forall\, s < u \bullet s \in traces(\mathcal{B}_{Ctr} \parallel_{\{ic,oc\}} BUFF^\infty_{IO})\}$
$=$ [Lemma 17 and Definition of $tr^\infty(u, (\mathcal{B}_{Ctr} \parallel_{\{ic,oc\}} BUFF^\infty_{IO})$ (18)]
$\{(u \setminus HC_{Unary}) ^\frown t \mid u \in \Sigma^\omega \wedge \neg\, (u \setminus HC_{Unary}) : \Sigma^\omega$
$\wedge\, \forall\, s < u \bullet s \in traces((\mathcal{B}_{Ctr} \parallel_{\{ic,oc\}} BUFF^\infty_{IO}) \wedge u \setminus HC_{Unary} : \Sigma^\omega\}$
$=$ [Property of Sets and Predicate Calculus]
$\{(u \setminus HC_{Unary}) ^\frown t \mid false\}$
$=$ [Property of Sets]
$\emptyset$

□

Theorem 5 establish the conditions under which feedback and reflexive infinite compositions are livelock-free.

In order to be able to perform further compositions using the resulting contracts in an efficient manner, we calculate the new *MIP* after every livelock-free composition. This information is stored in the contracts as metadata that aims at alleviating further verifications in our method for component composition.

4.4 Dealing with Metadata

The calculation of the *MIP*s of composed components can be based on the function proposed in [10] that calculates the traces of a parallel composition as the combination of the traces of each argument process, where the synchronised events are shared and all other events are interleaved. In our strategy, however, we are not concerned with the *MIP* generated by the interleaving of the *MIP*s because livelock can only be introduced by hiding events of a basic component.

For instance, using the merge from [10] to calculate the new *MIP*s of the interleaving composition Ctr_{Fork_1} $[|||]$ Ctr_{Fork_2}, we get all possible sequences resulting from merging $MIP(Fork_1)$ and $MIP(Fork_2)$:

$$\{\langle fk.1.1.up, fk.1.1.down, fk.2.2.up, fk.2.2.down\rangle$$
$$\langle fk.1.1.up, fk.2.2.up, fk.1.1.down, fk.2.2.down\rangle,$$
$$\langle fk.2.2.up, fk.2.2.up, fk.1.1.down, fk.1.1.down\rangle, \ldots\}$$

For any two minimum interaction patterns ip_1 and ip_2 from $MIP(Fork_1)$ and $MIP(Fork_2)$, respectively, this merge includes a large number of traces that communicate on the same channels from ip_1 and ip_2, which only differ in the order of the events. This order, however, is not relevant for our strategy because, using $\mathcal{BRIC}$, further compositions like, for instance, with a contract Ctr_3, will be made on a one channel to one channel basis. As a consequence, composing Ctr_3 with Ctr_1 $[|||]$ $Ctr2$ will be a communication between Ctr_3 with either Ctr_1 or Ctr_2. Based on this analysis, we provide a variation of the merge function from [10]. This optimisation is extremely relevant to the scalability of our approach.

Definition 14 (Optimised Trace Merge). *Let xs be a set of events, x and x' denote typical members of xs, and y denote a typical member of $\Sigma \setminus xs$. The optimised trace merge is defined as follows.*

$$\begin{array}{lr}
\langle\rangle \parallel_{xs}^{\langle s_0,t_0\rangle} \langle\rangle = \{\langle\rangle\} & (1)\\
\langle x\rangle \frown s \parallel_{xs}^{\langle s_0,t_0\rangle} \langle\rangle = \{u \mid u \in \langle x\rangle \frown s \parallel_{xs}^{\langle s_0,t_0\rangle} t_0\} & (2)\\
\langle\rangle \parallel_{xs}^{\langle s_0,t_0\rangle} \langle x\rangle \frown t = \{u \mid u \in s_0 \parallel_{xs}^{\langle s_0,t_0\rangle} \langle x\rangle \frown t\} & (3)\\
\langle y\rangle \frown s \parallel_{xs}^{\langle s_0,t_0\rangle} t = \{\langle y\rangle \frown u \mid u \in s \parallel_{xs}^{\langle s_0,t_0\rangle} t\} & (4)\\
s \parallel_{xs}^{\langle s_0,t_0\rangle} \langle y\rangle \frown t = \{\langle y\rangle \frown u \mid u \in s \parallel_{xs}^{\langle s_0,t_0\rangle} t\} & (5)\\
\langle x\rangle \frown s \parallel_{xs}^{\langle s_0,t_0\rangle} \langle x\rangle \frown t = \{u \mid u \in s \parallel_{xs}^{\langle s_0,t_0\rangle} t\} & (6)\\
\langle x\rangle \frown s \parallel_{xs}^{\langle s_0,t_0\rangle} \langle x'\rangle \frown t = \{\,\} & (7)
\end{array}$$

The differences between our definition for trace merging and that of [10] are: (1) Our merge function has the initial traces s_0 and t_0 as arguments. This allows us to merge n concatenations of s_0 with m concatenations of t_0; (2) In the cases in which one side is willing to perform a synchronisation event x and the other side has finished (lines 2 and 3), we "reset" the side that has finished, enforcing at least one synchronisation on x and decreasing the size of one of the sequences by at least one; (3) In the cases in which one side is willing to perform an independent event (lines 4 and 5), we do not take all possible combinations of permuting the independent events for the reasons previously explained; and (4) In the cases in which the synchronisation is feasible (line 6), our merge function does not include the synchronised event in the result because they are hidden after composition. We define the merge of interaction patterns as follows.

Definition 15 (MIP Merge). *Let Ctr_1 and Ctr_2 be two component contracts and ic and oc two communication channels in $\mathcal{C}_{Ctr_1}$ and $\mathcal{C}_{Ctr_2}$, respectively, and x a fresh channel name. The MIP merge function is defined as follows.*

$$\begin{array}{l}
MIPMerge(Ctr_1, Ctr_2, ic, oc) = \\
\quad \{s : MIP(\mathcal{B}_{Ctr_1}) \mid \{\!|\ ic\ |\!\} \cap \mathrm{ran}(s) = \emptyset\} \\
\quad \cup \{t : MIP(\mathcal{B}_{Ctr_2}) \mid \{\!|\ oc\ |\!\} \cap \mathrm{ran}(t) = \emptyset\} \\
\quad \cup \bigcup \left\{ \begin{array}{l}
s : MIP(\mathcal{B}_{Ctr_1});\ t : MIP(\mathcal{B}_{Ctr_2});\ sx, tx : \Sigma^* \\
\qquad \mid \{\!|\ ic\ |\!\} \cap \mathrm{ran}(s) \neq \emptyset \wedge \{\!|\ oc\ |\!\} \cap \mathrm{ran}(t) \neq \emptyset \\
\qquad\ \wedge\ sx \in ren(s, \{v : \mathrm{extensions}(ic) \bullet (ic.v, x.v)\}) \\
\qquad\ \wedge\ tx \in ren(t, \{v : \mathrm{extensions}(ic) \bullet (oc.v, x.v)\}) \\
\qquad \bullet\ sx \parallel_{\{\!|x|\!\}}^{\langle sx,tx\rangle} tx
\end{array} \right\}
\end{array}$$

The resulting merge contains all *MIP*s from $\mathcal{B}_{Ctr_1}$ and $\mathcal{B}_{Ctr_2}$ that do not have events on the connecting channels *ic* and *oc*, respectively. The remaining *MIP*s are merged using the optimised trace merge. Before the merge, however, the *MIP*s have to be unified on the events of the connecting channels. For that, we use a fresh channel name x and the function *ren* to replace for x references to *ic* and *oc* in $\mathcal{B}_{Ctr_1}$ and $\mathcal{B}_{Ctr_2}$, respectively. The function extensions(c) returns the values which will complete the channel yielding an event [10].

Next, the metadata calculation for the binary operators is as follows.

Proposition 2 (Binary Composition Metadata). *Let Ctr_1 and Ctr_2 be two component contracts and ic and oc two communication channels in $\mathcal{C}_{Ctr_1}$ and $\mathcal{C}_{Ctr_2}$, respectively. The MIP of the binary compositions are presented as follows.*

$$MIP(Ctr_1 \, [|||] \, Ctr_2) = MIP(\mathcal{B}_{Ctr_1}) \cup MIP(\mathcal{B}_{Ctr_2})$$
$$MIP(Ctr_1[ic \leftrightarrow oc]Ctr_2) = MIPMerge(Ctr_1, Ctr_2, ic, oc)$$

Finally, we formalise the metadata calculation for the unary compositions.

Proposition 3 (Unary Composition Metadata). *Let Ctr be a component contract and ic and oc two communication channels in $\mathcal{C}_{Ctr}$. The MIP of the unary compositions are presented as follows.*

$$MIP(Ctr[ic \hookrightarrow oc]) = \{s : MIP(\mathcal{B}_{Ctr}) \bullet s \setminus \{| \, ic, oc \, |\}\}$$
$$MIP(Ctr[ic \overset{-}{\hookrightarrow} oc]) = \{s : MIP(\mathcal{B}_{Ctr}) \bullet s \setminus \{| \, ic, oc \, |\}\}$$

The calculation of the resulting *MIP* for unary compositions simply removes both connecting channels from the original *MIP*s.

5 Evaluation

In this section, we demonstrate that our constructive approach to build livelock free models can be applied in practice to large systems involving several compositions. We have developed three case studies: the Milner's scheduler [6], which schedules a number of tasks and can be modelled as a ring of cell processes synchronised pairwisely, and two variations of the dining philosopher [10], a livelock-free version and a version in which we have deliberately included livelock. All case studies are developed using the the $\mathcal{BRIC}$ methodology, hence, we worked with asynchronous versions of these three case studies.

For each case study, we provide a comparative analysis of three scenarios: the global analysis of FDR2, the static analysis of SLAP, and our local analysis. In these case studies, we have used a dedicated server with an 8 core Intel(R) Core(TM) i7-2600K, 16 GB of RAM and 160GB of SSD in an Ubuntu system. The CSP scripts of these case studies can be found at http://goo.gl/mAZWXq.

Tables 1 and 2 summarise our results. The column N is the number of cells and philosophers for the Milner's scheduler and dining philosophers, respectively. The column # is the number of compositions, and the columns FDR2, SLAP and LLA present the time cost of the global analysis in FDR2, SLAP Static

Analysis (using BDD and SAT), and our local analysis (LLA). The * indicates one hour timeout and ** indicates memory overflow.

The results show that FDR2 and SLAP are unable to deal with large asynchronous configurations. On the other hand, our method verified, for instance, the absence of divergence for 10,000 philosophers and 10,000 forks (20,000 CSP processes and 39,988 $\mathcal{BRIC}^*$ compositions) in less than 4 minutes. This proved to be a very promising result in dealing with complex and large systems.

6 Conclusion

In this paper, we propose a correct-by-construction approach for ensuring livelock freedom in $\mathcal{BRIC}$ models built using four composition rules. The development of this strategy is based on the minimum sequences that represent patterns of interactions after which the system recurses. Considering only these finite sequences, we are able to locally assert livelock freedom before integrating components. Furthermore, we use metadata for storing information that alleviate verification conditions during component composition. To perform this analysis in $\mathcal{BRIC}$, we have provided a clear distinction of asynchronous compositions via finite and infinite buffers because the finiteness of the buffer is relevant for detecting the possibility of livelock in such systems.

We have used three case studies that demonstrate the scalability of our approach. For larger systems, the verification using FDR2 and SLAP may easily become costly and infeasible. On the other hand, our compositional livelock analysis seems promising as demonstrated in our case studies.

Our approach for local and compositional livelock analysis can still be improved. Parameters and non-tail recursion are not addressed here; they are, however, in our research agenda, which also includes additional case studies.

References

1. G. Beneken, U. Hammerschall, M. Broy, M. Cengarle, J. Jürjens, B. Rumpe, and M. Schoenmakers. Componentware - State of the Art 2003. October 2003.
2. Hans de Bruin. A grey-box approach to component composition. GCSE '99, pages 195–209. Springer-Verlag, 2000.
3. Madiel Filho, Marcel Oliveira, Augusto Sampaio, and Ana Cavalcanti. Local Livelock Analysis for Component-Based Models. Technical report, UFRN, 2016. http://goo.gl/zl1MQV.

Table 1. Results of the Livelock Analysis for the Milner's Scheduler in $\mathcal{BRIC}^*$.

N	#	FDR2	SLAP (BDD)	SLAP (SAT)	LLA
5	5	0.123s	0.045s	4.196s	0.177s
10	10	672.164s	0.128s	14.340s	0.218s
15	15	*	0.465s	29.862s	0.243s
100	100	*	2428.308s	**	0.559s
1,000	1,000	*	*	**	3.959s
3,000	3,000	*	*	**	7.578s

Table 2. Results of the Livelock Analysis for the Dining Philosophers in $\mathcal{BRIC}^*$.

		Livelock-free System				System with Livelock			
N	#	FDR2	SLAP (BDD)	SLAP (SAT)	LLA	FDR2	SLAP (BDD)	SLAP (SAT)	LLA
5	18	*	2.715s	14.135s	0.248s	*	1.638s	7.132s	0.246s
10	38	*	51.708s	383.884s	0.303s	*	26.259s	149.091s	0.297s
100	398	*	*	*	0.778s	*	*	**	0.769s
1,000	3,988	*	*	*	3.888s	*	*	**	3.431s
10,000	39,988	*	*	*	206.689s	*	*	**	185.209s

4. C.A.R. Hoare. *Communicating Sequential Processes.* Prentice-Hall, 1985.
5. Formal Systems Ltd. *FDR2: User Manual, version 2.94*, 2012.
6. R. Milner. *Communication and Concurrency.* Prentice-Hall, 1989.
7. Joel Ouaknine, Hristina Palikareva, A. W. Roscoe, and James Worrell. A static analysis framework for livelock freedom in CSP. *LMCS*, 9(3), 2013.
8. R. T. Ramos. *Systematic Development of Trustworthy Component-based Systems.* PhD thesis, Federal University of Pernambuco, 2011.
9. Rodrigo Ramos, Augusto Sampaio, and Alexandre Mota. Systematic development of trustworthy component systems. In *Formal Methods*, volume 5850 of *Lecture Notes in Computer Science*, pages 140–156. Springer, 2009.
10. A. W. Roscoe. *The Theory and Practice of Concurrency.* Prentice-Hall Series in Computer Science. Prentice-Hall, 1998.
11. A. W. Roscoe. The pursuit of buffer tolerance, 2005.
12. A. W. Roscoe. *Understanding Concurrent Systems.* Springer, 2010.
13. P. Soni and N. Ratti. Analysis of Component Composition Approaches. *International Journal of Computer Science and Communication Engineering*, 2(1), 2013.

7 Appendix A

In this section we present some auxiliary lemmas used in our proofs.

Lemma 1. *Let $Ctr_1\,{}_{\langle\rangle} \asymp {}_{\langle\rangle}\, Ctr_2$ be an asynchronous composition of Ctr_1 and Ctr_2 (Definition 3). Then, the projection of the behaviour of this composition over the set of communication channels, namely $\mathcal{B}_{Ctr_1\,{}_{\langle\rangle} \asymp {}_{\langle\rangle}\, Ctr_2} \upharpoonright \mathcal{C}_{Ctr_1\,{}_{\langle\rangle} \asymp {}_{\langle\rangle}\, Ctr_2}$, is described as follows:*

$$\left(\begin{array}{l} \mathcal{B}_{Ctr_1\,{}_{\langle\rangle} \asymp {}_{\langle\rangle}\, Ctr_2} \upharpoonright \mathcal{C}_{Ctr_1\,{}_{\langle\rangle} \asymp {}_{\langle\rangle}\, Ctr_2} \\ = \\ (\mathcal{B}_{Ctr_1} \setminus (\Sigma \setminus \mathcal{C}_{Ctr_1} \cup \mathcal{C}_{Ctr_2})) \mathbin{|||} (\mathcal{B}_{Ctr_2} \setminus (\Sigma \setminus \mathcal{C}_{Ctr_1} \cup \mathcal{C}_{Ctr_2})) \end{array}\right)$$

Proof

$$\mathcal{B}_{Ctr_1\,{}_{\langle\rangle} \asymp {}_{\langle\rangle}\, Ctr_2} \upharpoonright \mathcal{C}_{Ctr_1\,{}_{\langle\rangle} \asymp {}_{\langle\rangle}\, Ctr_2}$$

$=$ *[Definition 3]*

$$((\mathcal{B}_{Ctr_1} \mathbin{|||} \mathcal{B}_{Ctr_2}) \parallel_{\{\}} BUFF_{IO}^{\infty}) \upharpoonright ((\mathcal{C}_{Ctr_1} \cup \mathcal{C}_{Ctr_2}) \setminus \emptyset)$$

$=$ *[Property of Sets]*

$((\mathcal{B}_{Ctr_1} \interleave \mathcal{B}_{Ctr_2}) \parallel_{\{\}} BUFF_{IO}^{\infty}) \upharpoonright (\mathcal{C}_{Ctr_1} \cup \mathcal{C}_{Ctr_2})$

$=$ *[$BUFF_{IO}$ is not used]*

$((\mathcal{B}_{Ctr_1} \interleave \mathcal{B}_{Ctr_2}) \parallel_{\{\}} SKIP) \upharpoonright (\mathcal{C}_{Ctr_1} \cup \mathcal{C}_{Ctr_2})$

$=$ *[Law 5 (Parallel/Interleave)]*

$((\mathcal{B}_{Ctr_1} \interleave \mathcal{B}_{Ctr_2}) \interleave SKIP) \upharpoonright (\mathcal{C}_{Ctr_1} \cup \mathcal{C}_{Ctr_2})$

$=$ *[Law 4 (Interleave unit)]*

$(\mathcal{B}_{Ctr_1} \interleave \mathcal{B}_{Ctr_2}) \upharpoonright (\mathcal{C}_{Ctr_1} \cup \mathcal{C}_{Ctr_2})$

$=$ *[Definition 21 ($P \upharpoonright Ch$)]*

$(\mathcal{B}_{Ctr_1} \interleave \mathcal{B}_{Ctr_2}) \setminus (\Sigma \setminus \mathcal{C}_{Ctr_1} \cup \mathcal{C}_{Ctr_2})$

$=$ *[Law 2 (Hiding/Interleave distribution)]*

$(\mathcal{B}_{Ctr_1} \setminus (\Sigma \setminus \mathcal{C}_{Ctr_1} \cup \mathcal{C}_{Ctr_2})) \interleave (\mathcal{B}_{Ctr_2} \setminus (\Sigma \setminus \mathcal{C}_{Ctr_1} \cup \mathcal{C}_{Ctr_2}))$

□

Lemma 2. *Let P be a CSP process, and A and B two sets, such that, $A \subseteq B$. Then,*

$$divergences(P \setminus B) = \emptyset \Rightarrow divergences(P \setminus A) = \emptyset$$

Proof by contradiction

$\neg\,(divergences(P \setminus B) = \emptyset \Rightarrow divergences(P \setminus A) = \emptyset)$

$\Rightarrow$ *[Predicate Calculus]*

$divergences(P \setminus B) = \emptyset \wedge divergences(P \setminus A) \neq \emptyset$

$\Rightarrow$ *[$A \subseteq B \Rightarrow A \cup B = B$]*

$divergences(P \setminus (AUB)) = \emptyset \wedge divergences(P \setminus A) \neq \emptyset$

$\Rightarrow$ *[Law 1 (Hiding join)]*

$divergences((P \setminus A) \setminus B)) = \emptyset \wedge divergences(P \setminus A) \neq \emptyset$

$\Rightarrow$ *[Definition 16 $divergences(P \setminus X)$]*

$$\{(s \setminus B) ^\frown t \mid s \in divergences(P \setminus A)\} \cup \{(u \setminus B) ^\frown t \mid u : \Sigma^{\omega} \wedge \neg\,(u \setminus B) : \Sigma^{\omega} \wedge tr^{\infty}(u, P)\} = \emptyset \wedge divergences(P \setminus A) \neq \emptyset$$

$\Rightarrow$ *[Property of Sets]*

$\{(s \setminus B) ^\frown t \mid s \in divergences(P \setminus A)\} = \emptyset \wedge divergences(P \setminus A) \neq \emptyset$

$\Rightarrow$ *[Contradiction]*

false

Lemma 3. *Let $Ctr_1\,_{\langle ic\rangle} \asymp\ _{\langle oc\rangle}\, Ctr_2$ an asynchronous binary composition of Ctr_1 and Ctr_2 (Definition 3). Then, the projection of the behaviour of this composition over the set of communication channels, namely* $\mathcal{B}_{Ctr_1\,_{\langle ic\rangle} \asymp\ _{\langle oc\rangle}\, Ctr_2} \upharpoonright \mathcal{C}_{Ctr_1[ic \leftrightarrow oc]Ctr_2}$*, is described as follows:*

$$\begin{pmatrix} \mathcal{B}_{Ctr_1\,_{\langle ic\rangle} \asymp\ _{\langle oc\rangle}\, Ctr_2} \upharpoonright \mathcal{C}_{Ctr_1[ic \leftrightarrow oc]Ctr_2} \\ = \\ ((\mathcal{B}_{Ctr_1} \parallel\!| \mathcal{B}_{Ctr_2}) \parallel_{\{ic,oc\}} BUFF^{\infty}_{IO}) \setminus HC_{Bin} \end{pmatrix}$$

Proof

$\mathcal{B}_{Ctr_1\,_{\langle ic\rangle} \asymp\ _{\langle oc\rangle}\, Ctr_2} \upharpoonright \mathcal{C}_{Ctr_1[ic \leftrightarrow oc]Ctr_2}$

$=$ *[Definition 3]*

$((\mathcal{B}_{Ctr_1} \parallel\!| \mathcal{B}_{Ctr_2}) \parallel_{\{ic,oc\}} BUFF_{IO}F^{\infty}) \upharpoonright ((\mathcal{C}_{Ctr_1} \cup \mathcal{C}_{Ctr_2}) \setminus \{ic, oc\})$

$=$ *[Definition 21 ($P \upharpoonright Ch$)]*

$((\mathcal{B}_{Ctr_1} \parallel\!| \mathcal{B}_{Ctr_2}) \parallel_{\{ic,oc\}} BUFF^{\infty}_{IO}) \setminus (\Sigma \setminus (\mathcal{C}_{Ctr_1} \cup \mathcal{C}_{Ctr_2}) \setminus \{ic, oc\})$

$=$ *[Definition 19 (HC_{Bin})]*

$((\mathcal{B}_{Ctr_1} \parallel\!| \mathcal{B}_{Ctr_2}) \parallel_{\{ic,oc\}} BUFF^{\infty}_{IO}) \setminus HC_{Bin}$

□

Lemma 4. *Let P be a livelock-free process, and X a set of events. Then,*

$$\begin{pmatrix} divergences(P \setminus X) \\ = \\ \{(u \setminus X) ^\frown t \mid u : \Sigma^{\omega} \land \neg\, (u \setminus X) : \Sigma^{\omega} \land \forall\, s < u \bullet s \in traces(P)\} \end{pmatrix}$$

Proof

$divergences(P \setminus X)$

$=$ *[Definition 16 ($divergences(P \setminus X)$)]*

$\{(s \setminus X) ^\frown t \mid s \in divergences(P)\} \cup \{(u \setminus X) ^\frown t \mid u : \Sigma^{\omega} \land$
$\neg\, (u \setminus X) : \Sigma^{\omega} \land \forall\, s < u \bullet s \in traces_{\perp}(P)\}$

$=$ *[Definition 17 ($traces_{\perp}(P)$)]*

$\{(s \setminus X) ^\frown t \mid s \in divergences(P)\} \cup \{(u \setminus X) ^\frown t \mid u : \Sigma^{\omega} \land$
$\neg\, (u \setminus X) : \Sigma^{\omega} \land \forall\, s < u \bullet s \in traces(P) \cup\ divergences(P)\}$

$=$ *[P is a livelock-free process $\Rightarrow divergences(P) = \emptyset$]*

$\{(s \setminus X) ^\frown t \mid s \in \emptyset\} \cup \{(u \setminus X) ^\frown t \mid u : \Sigma^{\omega} \land \neg\, (u \setminus X) : \Sigma^{\omega}$
$\land \forall\, s < u \bullet s \in traces(P) \cup \emptyset\}$

$=$ *[Property of Sets]*

$\{(s \setminus X) ^\frown t \mid false\} \cup \{(u \setminus X) ^\frown t \mid u : \Sigma^{\omega} \land \neg\, (u \setminus X) : \Sigma^{\omega}$
$\land \forall\, s < u \bullet s \in traces(P) \cup\ \emptyset\}$

$=$ *[Property of Sets]*

$\emptyset \cup \{(u \setminus X) \frown t \mid u : \Sigma^\omega \wedge \neg\,(u \setminus X) : \Sigma^\omega$
$\wedge\, \forall\, s < u \bullet s \in traces(P) \cup\ \emptyset\}$

$=$ *[Property of Sets]*

$\{(u \setminus X) \frown t \mid u : \Sigma^\omega \wedge \neg\,(u \setminus X) : \Sigma^\omega \wedge \forall\, s < u \bullet s \in traces(P)\}$

□

Lemma 5. *Let $\mathcal{B}_P \interleave \mathcal{B}_Q$ be a CSP interleave composition of two I/O Processes, $\mathcal{B}_P$ and $\mathcal{B}_Q$, that are divergence-free and $BUFF^{\infty}_{IO}$ a livelock-free infinite buffer. Then, a CSP parallel composition of ($\mathcal{B}_P \interleave \mathcal{B}_Q$) and $BUFF^{\infty}_{IO}$, namely, $((\mathcal{B}_P \interleave \mathcal{B}_Q) \parallel_X BUFF^{\infty}_{IO})$, is divergence-free, formally*

$$divergences((\mathcal{B}_P \interleave \mathcal{B}_Q) \parallel_X BUFF^{\infty}_{IO}) = \emptyset$$

Proof

$divergences(((\mathcal{B}_P \interleave \mathcal{B}_Q) \parallel_X (BUFF^{\infty}_{IO})))$

$=$ *[Definition 16 ($divergences(P \parallel_X Q)$)]*

$\{u \frown v \mid \exists\, s \in traces(\mathcal{B}_P \interleave \mathcal{B}_Q), t \in traces(BUFF^{\infty}_{IO}) \bullet$
$u \in (s \parallel t) \cap\ \Sigma^* \wedge (s \in divergences(\mathcal{B}_P \interleave \mathcal{B}_Q) \vee t \in divergences(BUFF^{\infty}_{IO}))\}$

$=$ *[Lemma 6]*

$\{u \frown v \mid \exists\, s \in traces(\mathcal{B}_P \interleave \mathcal{B}_Q), t \in traces(BUFF^{\infty}_{IO}) \bullet$
$u \in (s \parallel t) \cap\ \Sigma^* \wedge (s \in \emptyset \vee t \in divergences(BUFF^{\infty}_{IO}))\}$

$=$ *[$divergences(BUFF^{\infty}_{IO}) = \emptyset$]*

$\{u \frown v \mid \exists\, s \in traces(\mathcal{B}_P \interleave \mathcal{B}_Q), t \in traces(BUFF^{\infty}_{IO}) \bullet$
$u \in (s \parallel t) \cap\ \Sigma^* \wedge (s \in \emptyset \vee t \in \emptyset)\}$

$=$ *[Property of Sets and Predicate Calculus]*

$\{u \frown v \mid false\}$

$=$ *[Property of Sets]*

$\emptyset$

□

Lemma 6. *Let $\mathcal{B}_P$ and $\mathcal{B}_Q$ be two I/O Processes. Then, a CSP interleave composition of $\mathcal{B}_P$ and $\mathcal{B}_Q$, ($\mathcal{B}_P \interleave \mathcal{B}_Q$), is divergence-free, formally:*

$$divergences(\mathcal{B}_P \interleave \mathcal{B}_Q) = \emptyset$$

Proof

$divergences(\mathcal{B}_P \interleave \mathcal{B}_Q)$

$=$ *[Definition 16($divergences(\mathcal{B}_P \interleave \mathcal{B}_Q)$)]*

$\{u \frown v \mid \exists\, s \in traces(\mathcal{B}_P), t \in traces(\mathcal{B}_Q) \bullet$
$\quad u \in (s \parallel t) \cap \Sigma^* \wedge (s \in divergences(\mathcal{B}_P) \vee t \in divergences(\mathcal{B}_Q))\}$

$=$ *[$divergences(\mathcal{B}_P) = \emptyset \wedge divergences(\mathcal{B}_Q) = \emptyset$ by Definition 2]*

$\{u \frown v \mid \exists\, s \in traces(\mathcal{B}_P), t \in traces(\mathcal{B}_Q) \bullet$
$\quad u \in (s \parallel t) \cap \Sigma^* \wedge (s \in \emptyset \vee t \in \emptyset))\}$

$=$ *[Property of Sets and Predicate Calculus]*

$\{u \frown v \mid false\}$

$=$ *[Property of Sets]*

$\emptyset$

□

Lemma 7. *Let Ctr_1 and Ctr_2 be two livelock-free component contracts, ic a communication channel in $Allowed_{Bric}(Ctr_1)$, and oc a communication channel in $Allowed_{Bric}(Ctr_2)$. Then,*

$$\forall\, u_c : \Sigma^\omega \mid tr^\infty(u_c, (\mathcal{B}_{Ctr_1} \interleave \mathcal{B}_{Ctr_2}) \parallel_{\{ic,oc\}} BUFF^\infty_{IO}) \bullet (u_c \setminus HC_{Bin} : \Sigma^\omega)$$

provided $ic \in Allowed_{Bric}(Ctr_1) \wedge oc \in Allowed_{Bric}(Ctr_2)$

Proof

$tr^\infty(u_c, (\mathcal{B}_{Ctr_1} \interleave \mathcal{B}_{Ctr_2}) \parallel_{\{ic,oc\}} BUFF^\infty_{IO})$

$\Rightarrow$ [Lemma 8]

$\exists\, u : \Sigma^\omega \mid tr^\infty(u, \mathcal{B}_{Ctr_1}) \vee tr^\infty(u, \mathcal{B}_{Ctr_2}) \bullet u\ in^\infty\ u_c$

$\equiv$ [Predicate Calculus]

$\exists\, u : \Sigma^\omega \mid tr^\infty(u, \mathcal{B}_{Ctr_1}) \bullet u\ in^\infty\ u_c$
$\vee\ \exists\, u : \Sigma^\omega \mid tr^\infty(u, \mathcal{B}_{Ctr_2}) \bullet u\ in^\infty\ u_c$

$\equiv$ [Property of hiding for sequences]

$\exists\, u : \Sigma^\omega \mid tr^\infty(u, \mathcal{B}_{Ctr_1}) \bullet u \setminus HC_{Bin}\ in^\infty\ u_c \setminus HC_{Bin}$
$\vee\ \exists\, u : \Sigma^\omega \mid tr^\infty(u, \mathcal{B}_{Ctr_2}) \bullet u \setminus HC_{Bin}\ in^\infty\ u_c \setminus HC_{Bin}$

$\Rightarrow$ [Lemma 13 and proviso]

$\exists\, u : \Sigma^\omega \mid tr^\infty(u, \mathcal{B}_{Ctr_1}) \bullet u \setminus HC_{Bin}\ in^\infty\ u_c \setminus HC_{Bin} \wedge (u \setminus HC_{Bin} : \Sigma^\omega)$
$\vee\ \exists\, u : \Sigma^\omega \mid tr^\infty(u, \mathcal{B}_{Ctr_2}) \bullet u \setminus HC_{Bin}\ in^\infty\ u_c \setminus HC_{Bin} \wedge (u \setminus HC_{Bin} : \Sigma^\omega)$

$\Rightarrow$ [Property of Infinite Sequences ($s\ in^\infty\ t \wedge s : \Sigma^\omega \Rightarrow t : \Sigma^\omega$)]

$\exists\, u : \Sigma^\omega \mid tr^\infty(u, \mathcal{B}_{Ctr_1}) \bullet u_c \setminus HC_{Bin} : \Sigma^\omega$
$\vee\ \exists\, u : \Sigma^\omega \mid tr^\infty(u, \mathcal{B}_{Ctr_2}) \bullet u_c \setminus HC_{Bin} : \Sigma^\omega$

$\equiv$ [Predicate Calculus]

$u_c \setminus HC_{Bin} : \Sigma^\omega$

□

Lemma 8.

$$\forall u_c : \Sigma^\omega \mid tr^\infty(u_c, (\mathcal{B}_{Ctr_1} \interleave \mathcal{B}_{Ctr_2}) \parallel_{\{ic,oc\}} BUFF_{IO}^\infty) \bullet$$
$$\exists u : \Sigma^\omega \mid tr^\infty(u, \mathcal{B}_{Ctr_1}) \lor tr^\infty(u, \mathcal{B}_{Ctr_2}) \bullet u \; in^\infty \; u_c$$

Proof

$\forall u_c : \Sigma^\omega \mid tr^\infty(u_c, (\mathcal{B}_{Ctr_1} \interleave \mathcal{B}_{Ctr_2}) \parallel_{\{ic,oc\}} BUFF_{IO}^\infty) \bullet$

$\quad \exists u : \Sigma^\omega \mid tr^\infty(u, \mathcal{B}_{Ctr_1}) \lor tr^\infty(u, \mathcal{B}_{Ctr_2}) \bullet u \; in^\infty \; u_c$

$\equiv$ [Predicate Calculus]

$\forall u_c : \Sigma^\omega \bullet tr^\infty(u_c, (\mathcal{B}_{Ctr_1} \interleave \mathcal{B}_{Ctr_2}) \parallel_{\{ic,oc\}} BUFF_{IO}^\infty) \Rightarrow$

$\quad \exists u : \Sigma^\omega \mid tr^\infty(u, \mathcal{B}_{Ctr_1}) \lor tr^\infty(u, \mathcal{B}_{Ctr_2}) \bullet u \; in^\infty \; u_c$

We start from the left-hand side of the predicate and get the right-hand side.

$tr^\infty(u_c, (\mathcal{B}_{Ctr_1} \interleave \mathcal{B}_{Ctr_2}) \parallel_{\{ic,oc\}} BUFF_{IO}^\infty)$

$\equiv$ [Definition of tr^∞ (18)]

$\forall s < u_c \bullet s \in traces((\mathcal{B}_{Ctr_1} \interleave \mathcal{B}_{Ctr_2}) \parallel_{\{ic,oc\}} BUFF_{IO}^\infty)$

$\Rightarrow$ [Lemma 9]

$\forall s < u_c \bullet s \in traces(\mathcal{B}_{Ctr_1} \interleave \mathcal{B}_{Ctr_2})$

$\Rightarrow$ [Generalisation of Lemma 12 for infinite traces]

$\exists u : \Sigma^\omega \bullet (\forall s < u \bullet s \in traces(\mathcal{B}_{Ctr_1})) \land u \; in^\infty \; u_c$

$\lor \exists u : \Sigma^\omega \bullet (\forall s < u \bullet s \in traces(\mathcal{B}_{Ctr_2})) \land u \; in^\infty \; u_c$

$\equiv$ [Predicate Calculus]

$\exists u : \Sigma^\omega \bullet ((\forall s < u \bullet s \in traces(\mathcal{B}_{Ctr_1})) \land u \; in^\infty \; u_c)$

$\quad \lor ((\forall s < u \bullet s \in traces(\mathcal{B}_{Ctr_2})) \land u \; in^\infty \; u_c)$

$\equiv$ [Definition of tr^∞ (18)]

$\exists u : \Sigma^\omega \bullet (tr^\infty(u, \mathcal{B}_{Ctr_1}) \land u \; in^\infty \; u_c) \lor (tr^\infty(u, \mathcal{B}_{Ctr_2}) \land u \; in^\infty \; u_c)$

$\equiv$ [Predicate Calculus]

$\exists u : \Sigma^\omega \bullet (tr^\infty(u, \mathcal{B}_{Ctr_1}) \lor tr^\infty(u, \mathcal{B}_{Ctr_2})) \land u \; in^\infty \; u_c$

$\equiv$ [Predicate Calculus]

$\exists u : \Sigma^\omega \mid tr^\infty(u, \mathcal{B}_{Ctr_1}) \lor tr^\infty(u, \mathcal{B}_{Ctr_2}) \bullet u \; in^\infty \; u_c$

□

Lemma 9.

Let P be an I/O Process, with alphabet A, $BUFF_{IO}^{\infty}$ an infinite buffer.

$$\begin{array}{l} traces(P \parallel_X BUFF_{IO}^{\infty}) \\ \subseteq \\ traces(P) \end{array}$$

Proof

$$\begin{array}{lr}
traces(P \parallel_X BUFF_{IO}^{\infty}) \subseteq traces(P) & \\
= & \textit{[Definition of traces}(P_1 \parallel_X P_2)\textit{]} \\
\bigcup\{z \parallel_X t \mid z \in traces(P) \land t \in traces(BUFF_{IO}^{\infty})\} \subseteq traces(P) & \\
= & \textit{[Property of Set Comprehension]} \\
\forall s \bullet s \in \bigcup\{z \parallel_X t \mid z \in traces(P) \land t \in traces(BUFF_{IO}^{\infty})\} \Rightarrow s \in traces(P) & \\
= & \textit{[Predicate Calculus]} \\
\forall s \bullet (\exists z : traces(P);\ t : traces(BUFF_{IO}^{\infty}) \bullet s \in z \parallel_X t) \Rightarrow s \in traces(P) & \\
= & \textit{[Predicate Calculus]} \\
\forall s \bullet \neg\, (\exists z : traces(P);\ t : traces(BUFF_{IO}^{\infty}) \bullet s \in z \parallel_X t) \lor s \in traces(P) & \\
= & \textit{[Predicate Calculus]} \\
\forall s \bullet (\forall z : traces(P);\ t : traces(BUFF_{IO}^{\infty}) \bullet \neg\, (s \in z \parallel_X t)) \lor s \in traces(P) & \\
= & \textit{[Predicate Calculus]} \\
\forall s;\ z : traces(P);\ t : traces(BUFF_{IO}^{\infty}) \bullet s \in z \parallel_X t \Rightarrow s \in traces(P) & \\
= & \textit{[Lemma 10]} \\
\forall s;\ z : traces(P);\ t : traces(BUFF_{IO}^{\infty}) \bullet s \in \emptyset \lor s \in \{z\} \Rightarrow s \in traces(P) & \\
= & \textit{[Property of Sets and Predicate Calculus]} \\
\forall s;\ z : traces(P);\ t : traces(BUFF_{IO}^{\infty}) \bullet \mathit{false} \lor s \in \{z\} \Rightarrow s \in traces(P) & \\
= & \textit{[Predicate Calculus]} \\
\forall s;\ z : traces(P) \bullet s \in \{z\} \Rightarrow s \in traces(P) & \\
= & \textit{[Predicate Calculus]} \\
\forall s;\ z : traces(P) \bullet s = z \Rightarrow s \in traces(P) & \\
= & \textit{[Law one-point rule]} \\
\forall z : traces(P) \bullet z \in traces(P) & \\
= & \textit{[Predicate Calculus]} \\
\mathit{true} &
\end{array}$$

□

Lemma 10.
Let P be an I/O Process, with alphabet A, $BF^{\infty} = BUFF_{IO}[in \leftarrow x_1, out \leftarrow x_2]$, with $X = \{x_1, x_2\} \subset A$. Then,

$$\forall z : traces(P);\ t : traces(BF^{\infty}) \bullet (z \parallel_X t = \emptyset \lor z \parallel_X t = \{z\})$$

Proof by induction on z:
- *Base case:* $z = \langle\rangle$

$(\langle\rangle \parallel_X t = \emptyset) \vee (\langle\rangle \parallel_X t = \{\langle\rangle\})$
$=$ *[Calculation of* $z \parallel_X t \wedge \mathrm{ran}(t) \subseteq X$*]*
$(\emptyset = \emptyset) \vee (\langle\rangle \parallel_X t = \{\langle\rangle\})$
$=$ *[Predicate calculus]*
$true \vee (\langle\rangle \parallel_X t = \{\langle\rangle\}$
$=$ *[Predicate calculus]*
$true$

Induction step

– *Induction Hypothesis (1):*

$$\forall t : traces(BF^{\infty}) \bullet (z \parallel_X t = \emptyset) \vee (z \parallel_X t = \{z\})$$

We need to prove:
$(\langle m\rangle \frown z \parallel_X t = \emptyset) \vee (\langle m\rangle \frown z \parallel_X t = \{\langle m\rangle \frown z\})$
Proof by induction on t
– *Base case:* $t = \langle\rangle$
.1 - $m \in X$

$(\langle m\rangle \frown z \parallel_X \langle\rangle = \emptyset) \vee (\langle m\rangle \frown z \parallel_X \langle\rangle = \{\langle m\rangle \frown z\})$
$=$ *[Calculation of* $z \parallel_X t \wedge m \in X$*]*
$(\emptyset = \emptyset) \vee (\langle m\rangle \frown z \parallel_X \langle\rangle = \{\langle m\rangle \frown z\})$
$=$ *[Predicate calculus]*
$true \vee (\langle m\rangle \frown z \parallel_X \langle\rangle = \{\langle m\rangle \frown z\})$
$=$ *[Predicate calculus]*
$true$

.2 - $m \notin X$

$(\langle m\rangle \frown z \parallel_X \langle\rangle = \emptyset) \vee (\langle m\rangle \frown z \parallel_X \langle\rangle = \{\langle m\rangle \frown z\})$
$=$
[Calculation of $z \parallel_X t \wedge m \notin X$*]*
$(\{\langle m\rangle \frown u \mid u \in z \parallel_X \langle\rangle\} = \emptyset) \vee (\{\langle m\rangle \frown u \mid u \in z \parallel_X \langle\rangle\} = \{\langle m\rangle \frown z\})$
$=$ *[Lemma 11, induction hypothesis, and* $\langle\rangle \in traces(BF^{\infty})$*]*
$true$

Lemma 11. *Let be* $z : \Sigma^*$ *such that* $z \parallel_X \langle\rangle \vee z \parallel_X \langle\rangle = \{z\}$ *. Then;*

$$\{\langle m\rangle \frown u \mid u \in z \parallel_X \langle\rangle\} = \emptyset \vee \{\langle m\rangle \frown u \mid u \in z \parallel_X \langle\rangle\} = \{\langle m\rangle \frown z\}$$

provided $z \parallel t = \emptyset \vee z \parallel t = \{z\}$
Base case: $z = \langle\rangle$

$$\{\langle m\rangle \frown u \mid u \in z \parallel_X \langle\rangle\} = \emptyset \vee \{\langle m\rangle \frown u \mid u \in z \parallel_X \langle\rangle\} = \{\langle m\rangle \frown z\}$$
$= \qquad$ *[$z = \langle\rangle$]*
$$\{\langle m\rangle \frown u \mid u \in \langle\rangle \parallel_X \langle\rangle\} = \emptyset \vee \{\langle m\rangle \frown u \mid u \in \langle\rangle \parallel_X \langle\rangle\} = \{\langle m\rangle \frown \langle\rangle\}$$
$= \qquad$ *[Calculation of $z \parallel t$]*
$$\{\langle m\rangle \frown u \mid u \in \{\langle\rangle\}\} = \emptyset \vee \{\langle m\rangle \frown u \mid u \in \{\langle\rangle\}\} = \{\langle m\rangle\}$$
$= \qquad$ *[Predicate calculus]*
$$(\{\langle m\rangle \frown u \mid u \in \{\langle\rangle\}\} = \emptyset) \vee (\{\langle m\rangle\} = \{\langle m\rangle\})$$
$= \qquad$ *[Predicate Calculus]*
$$(\{\langle m\rangle \frown u \mid u \in \{\langle\rangle\}\} = \emptyset) \vee \mathit{true}$$
$= \qquad$ *[Predicate Calculus]*
$$\mathit{true}$$

Induction Step
Induction Hypothesis:

$$\{\langle m\rangle \frown u \mid u \in z \parallel_X \langle\rangle\} = \emptyset \vee \{\langle m\rangle \frown u \mid u \in z \parallel_X \langle\rangle\} = \{\langle m\rangle \frown z\}$$

We need to prove:

$$\{\langle m\rangle \frown u \mid u \in \langle m'\rangle \frown z \parallel_X \langle\rangle\} = \emptyset \vee \{\langle m\rangle \frown u \mid u \in \langle m'\rangle \frown z \parallel_X \langle\rangle\} = \{\langle m\rangle \frown \langle m'\rangle \frown z\}$$

Case analysis:
First case: $m' \in X$

$$\{\langle m\rangle \frown u \mid u \in \langle m'\rangle \frown z \parallel_X \langle\rangle\} = \emptyset \vee \{\langle m\rangle \frown u \mid u \in \langle m'\rangle \frown z \parallel_X \langle\rangle\} = \{\langle m\rangle \frown \langle m'\rangle \frown z\}$$
$= \qquad$ *[Calculation of $z \parallel_X t \wedge m \in X \wedge m' \in X$]*
$$\{\langle m\rangle \frown u \mid u \in \emptyset\} = \emptyset \vee \{\langle m\rangle \frown u \mid u \in \emptyset\} = \{\langle m\rangle \frown \langle m'\rangle \frown z\}$$
$= \qquad$ *[Property of Sets and Predicate Calculus]*
$$\{\langle m\rangle \frown u \mid \mathit{false}\} = \emptyset \vee \{\langle m\rangle \frown u \mid \mathit{false}\} = \{\langle m\rangle \frown \langle m'\rangle \frown z\}$$
$= \qquad$ *[Predicate calculus]*
$$\emptyset = \emptyset \vee \{\langle m\rangle \frown u \mid \mathit{false}\} = \{\langle m\rangle \frown \langle m'\rangle \frown z\}$$
$= \qquad$ *[Predicate calculus]*
$$\mathit{true} \vee \{\langle m\rangle \frown u \mid \mathit{false}\} = \{\langle m\rangle \frown \langle m'\rangle \frown z\}$$
$= \qquad$ *[Predicate calculus]*

true

Second case: $m' \notin X$

$\{\langle m\rangle \frown u \mid u \in \langle m'\rangle \frown z \parallel_X \langle\rangle\} = \emptyset \vee \{\langle m\rangle \frown u \mid$
$u \in \langle m'\rangle \frown z \parallel_X \langle\rangle\} = \{\langle m\rangle \frown \langle m'\rangle \frown z\}$
$=$ *[Calculation of* $z \parallel_X t \wedge m' \notin X$*]*
$\{\langle m\rangle \frown u \mid u \in \{\langle m'\rangle \frown u \mid u \in z \parallel_X \langle\rangle\}\} = \emptyset$
$\vee \{\langle m\rangle \frown u \mid u \in \{\langle m'\rangle \frown u \mid u \in z \parallel_X \langle\rangle\}\} = \{\langle m\rangle \frown \langle m'\rangle \frown z\}$

Case 1: $z \parallel \langle\rangle = \emptyset$ *proviso*

$\{\langle m\rangle \frown u \mid u \in \{\langle m'\rangle \frown u \mid u \in z \parallel_X \langle\rangle\}\} = \emptyset$
$\vee \{\langle m\rangle \frown u \mid u \in \{\langle m'\rangle \frown u \mid u \in z \parallel_X \langle\rangle\}\} = \{\langle m\rangle \frown \langle m'\rangle \frown z\}$
$=$ *[*$z \parallel t = \emptyset$*]*
$\{\langle m\rangle \frown u \mid u \in \{\langle m'\rangle \frown u \mid u \in \emptyset\}\} = \emptyset \vee \{\langle m\rangle \frown u \mid$
$u \in \{\langle m'\rangle \frown u \mid u \in \emptyset\}\} = \{\langle m\rangle \frown \langle m'\rangle \frown z\}$
$=$ *[Predicate Calculus and Property of Sets]*
$\{\langle m\rangle \frown u \mid u \in \mathit{false}\} = \emptyset \vee \{\langle m\rangle \frown u \mid u \in \mathit{false}\} = \{\langle m\rangle \frown \langle m'\rangle \frown z\}$
$=$ *[Predicate Calculus]*
$\emptyset = \emptyset \vee \{\langle m\rangle \frown u \mid u \in \mathit{false}\} = \{\langle m\rangle \frown \langle m'\rangle \frown z\}$
$=$ *[Predicate Calculus]*
$\mathit{true} \vee \{\langle m\rangle \frown u \mid u \in \mathit{false}\} = \{\langle m\rangle \frown \langle m'\rangle \frown z\}$
$=$ *[Predicate Calculus]*
true

Case 2: $z \parallel_X \langle\rangle = \{z\}$ *proviso*

$\{\langle m\rangle \frown u \mid u \in \{\langle m'\rangle \frown u \mid u \in z \parallel_X \langle\rangle\}\} = \emptyset$
$\vee \{\langle m\rangle \frown u \mid u \in \{\langle m'\rangle \frown u \mid u \in z \parallel_X \langle\rangle\}\} = \{\langle m\rangle \frown \langle m'\rangle \frown z\}$
$=$ *[*$z \parallel_X \langle\rangle = \{z\}$*]*
$\{\langle m\rangle \frown u \mid u \in \{\langle m'\rangle \frown u \mid u \in \{z\}\}\} = \emptyset$
$\vee \{\langle m\rangle \frown u \mid u \in \{\langle m'\rangle \frown u \mid u \in \{z\}\} = \{\langle m\rangle \frown \langle m'\rangle \frown z\}$
$=$ *[Predicate Calculus]*
$\{\langle m\rangle \frown u \mid u \in \{\langle m'\rangle \frown z\}\}\} = \emptyset$
$\vee \{\langle m\rangle \frown u \mid u \in \{\langle m'\rangle \frown z\}\} = \{\langle m\rangle \frown \langle m'\rangle \frown z\}$
$=$ *[Predicate Calculus]*
$\{\langle m\rangle \frown u \mid u \in \{\langle m'\rangle \frown z\}\}\} = \emptyset$
$\vee \{\langle m\rangle \frown \langle m'\rangle \frown z\}\} = \{\langle m\rangle \frown \langle m'\rangle \frown z\}$

$=$ *[Predicate Calculus]*

$\{\langle m\rangle \frown u \mid u \in \{\langle m'\rangle \frown z\}\}\} = \emptyset$

$\vee$ *true*

$=$ *[Predicate Calculus]*

true

Induction step

– *Induction Hypothesis (2):*

$$(\langle m\rangle \frown z \parallel_X t = \emptyset) \vee (\langle m\rangle \frown z \parallel_X t = \{\langle m\rangle \frown z\})$$

We need to prove

$$(\langle m\rangle \frown z \parallel_X \langle m'\rangle \frown t = \emptyset) \vee (\langle m\rangle \frown z \parallel_X \langle m'\rangle \frown t = \{\langle m\rangle \frown z\})$$

Case analysis:
First case: $\langle m\rangle \frown z \parallel_X \langle m'\rangle \frown t \wedge m \in X \wedge m' \in X \wedge m \neq m'$

$(\langle m\rangle \frown z \parallel_X \langle m'\rangle \frown t = \emptyset) \vee (\langle m\rangle \frown z \parallel_X \langle m'\rangle \frown t = \{\langle m\rangle \frown z\})$

$=$ *[Calculation of* $z \parallel_X t \wedge m \in X \wedge m' \in X \wedge m \neq m'$*]*

$(\emptyset = \emptyset) \vee (\langle m\rangle \frown z \parallel_X \langle m'\rangle \frown t = \{\langle m\rangle \frown z\})$

$=$ *[Predicate calculus]*

true $\vee (\langle m\rangle \frown z \parallel_X \langle m'\rangle \frown t = \{\langle m\rangle \frown z\})$

$=$ *[Predicate calculus]*

true

Second case: $\langle m\rangle \frown z \parallel_X \langle m'\rangle \frown t \wedge m \in X \wedge m' \in X \wedge m = m'$

$(\langle m\rangle \frown z \parallel_X \langle m'\rangle \frown t = \emptyset) \vee (\langle m\rangle \frown z \parallel_X \langle m'\rangle \frown t = \{\langle m\rangle \frown z\})$

$=$ *[Calculation of* $z \parallel_X t \wedge m = m' \wedge m \in X$*]*

$(\{\langle m\rangle \frown u \mid u \in z \parallel_X t\} = \emptyset) \vee \{\langle m\rangle \frown u \mid u \in z \parallel_X t\} = \{\langle m\rangle \frown z\}\}$

Case 1: $z \parallel_X t = \emptyset$ *(IH I)*

$(\{\langle m\rangle \frown u \mid u \in z \parallel_X t\} = \emptyset) \vee \{\langle m\rangle \frown u \mid u \in z \parallel_X t\} = \{\langle m\rangle \frown z\}\}$

$=$ *[*$z \parallel_X t = \emptyset$ *(IH I)]*

$(\{\langle m\rangle \frown u \mid u \in \emptyset\} = \emptyset) \vee (\{\langle m\rangle \frown u \mid u \in \emptyset\} = \{\langle m\rangle \frown z\})$

$=$ *[Property of Sets and Predicate calculus]*

$(\{\langle m\rangle \frown u \mid \mathit{false}\} = \emptyset) \vee (\{\langle m\rangle \frown u \mid u \in \emptyset\} = \{\langle m\rangle \frown z\})$

$=$ *[Property of Sets]*

$(\emptyset = \emptyset) \vee (\{\langle m\rangle \frown u \mid u \in \emptyset\} = \{\langle m\rangle \frown z\})$

$=$ *[Predicate calculus]*

$true \vee (\{\langle m\rangle \frown u \mid u \in \emptyset\} = \{\langle m\rangle \frown z\})$

$=$ *[Predicate calculus]*

$true$

Case 2: $z \parallel_X t = \{z\}$ *(IH I)*

$(\langle m\rangle \frown z \parallel_X \langle m'\rangle \frown t = \emptyset) \vee (\langle m\rangle \frown z \parallel_X \langle m'\rangle \frown t = \{\langle m\rangle \frown z\})$

$=$ *[$z \parallel_X t = \{z\}$ (IH I)]*

$(\{\langle m\rangle \frown u \mid u \in \{z\}\} = \emptyset) \vee (\{\langle m\rangle \frown u \mid u \in \{z\}\} = \{\langle m\rangle \frown z\})$

$=$ *[Predicate calculus]*

$\{\langle m\rangle \frown u \mid u \in \{z\}\} = \emptyset \vee \{\langle m\rangle \frown z\} = \{\langle m\rangle \frown z\}$

$=$ *[Predicate calculus]*

$\{\langle m\rangle \frown u \mid u \in \{z\}\} = \emptyset \vee true$

$=$ *[Predicate calculus]*

$true$

Third case: $\langle m\rangle \frown z \parallel_X \langle m'\rangle \frown t \wedge m \notin X \wedge m' \in X$

$\langle m\rangle \frown z \parallel_X \langle m'\rangle \frown t = \emptyset \vee \langle m\rangle \frown z \parallel_X \langle m'\rangle \frown t = \{\langle m\rangle \frown z\}$

$=$ *[Calculation of $z \parallel_X t \wedge m \notin X \wedge m' \in X$]*

$\{\langle m\rangle \frown u \mid u \in z \parallel_X \langle m'\rangle \frown t\} = \emptyset \vee \{\langle m\rangle \frown u \mid$
$u \in z \parallel_X \langle m'\rangle \frown t\} = \{\langle m\rangle \frown z\}\}$

Case 1: $z \parallel_X \langle m'\rangle \frown t = \emptyset$ *(IH I)*

$\{\langle m\rangle \frown u \mid u \in z \parallel_X \langle m'\rangle \frown t\} = \emptyset \vee \{\langle m\rangle \frown u \mid$
$u \in z \parallel_X \langle m'\rangle \frown t\} = \{\langle m\rangle \frown z\}\}$

$=$ *[$z \parallel_X \langle m'\rangle \frown t = \emptyset$ (IH I)]*

$\{\langle m\rangle \frown u \mid u \in \emptyset\} = \emptyset \vee \{\langle m\rangle \frown u \mid u \in \emptyset\} = \{\langle m\rangle \frown z\}$

$=$ *[Property of Ssets and Predicate Calculus]*

$\{\langle m\rangle \frown u \mid false\} = \emptyset \vee \{\langle m\rangle \frown u \mid u \in \emptyset\} = \{\langle m\rangle \frown z\}$

$=$ *[Property of Sets]*

$\emptyset = \emptyset \vee \{\langle m\rangle \frown u \mid u \in \emptyset\} = \{\langle m\rangle \frown z\}$

$=$ *[Predicate calculus]*

$true \vee \{\langle m\rangle \frown u \mid u \in \emptyset\} = \{\langle m\rangle \frown z\}$

$=$ *[Predicate calculus]*

$true$

Case 2: $z \parallel_X \langle m'\rangle \frown t = \{z\}$ *(IH I)*

$\{\langle m\rangle \frown u \mid u \in z \parallel_X \langle m'\rangle \frown t\} = \emptyset \vee \{\langle m\rangle \frown u \mid$
$u \in z \parallel_X \langle m'\rangle \frown t\} = \{\langle m\rangle \frown z\}\}$
$=$ *[z $\parallel_X$ $\langle m'\rangle \frown t = \{z\}$ (IH I)]*
$\{\langle m\rangle \frown u \mid u \in \{z\}\} = \emptyset) \vee (\{\langle m\rangle \frown u \mid$
$u \in \{z\}\} = \{\langle m\rangle \frown z\}$
$=$ *[Predicate calculus]*
$\{\langle m\rangle \frown u \mid u \in \{z\}\} = \emptyset \vee \{\langle m\rangle \frown z\} = \{\langle m\rangle \frown z\}$
$=$ *[Predicate calculus]*
$\{\langle m\rangle \frown u \mid u \in \{z\}\} = \emptyset \vee \mathit{true}$
$=$ *[Predicate calculus]*
true

Lemma 12.

$\forall\, u_c : \Sigma^* \bullet$
$\quad u_c \in \mathit{traces}(\mathcal{B}_{Ctr_1} \interleave \mathcal{B}_{Ctr_2})$
$\quad \Rightarrow$
$\quad \exists\, u : \mathit{traces}(\mathcal{B}_{Ctr_1}) \bullet u \; \mathit{in} \; u_c \vee \exists\, u : \mathit{traces}(\mathcal{B}_{Ctr_2}) \bullet u \; \mathit{in} \; u_c$

Proof By induction on u_c
• *Base case: $u_c = \langle\rangle$*
We start from the left-hand side of the predicate and get the right-hand side.

$\langle\rangle \in \mathit{traces}(\mathcal{B}_{Ctr_1} \interleave \mathcal{B}_{Ctr_2})$
$\Rightarrow$ *[Property of traces(P)]*
$\exists\, u : \mathit{traces}(\mathcal{B}_{Ctr_1}) \bullet u = \langle\rangle \vee \exists\, u : \mathit{traces}(\mathcal{B}_{Ctr_2}) \bullet u = \langle\rangle$
$\equiv$ *[Property of Subsequence and Predicate Calculus]*
$\exists\, u : \mathit{traces}(\mathcal{B}_{Ctr_1}) \bullet u \; \mathit{in} \; \langle\rangle \vee \exists\, u : \mathit{traces}(\mathcal{B}_{Ctr_2}) \bullet u \; \mathit{in} \; \langle\rangle$

Induction step
• *Induction Hypothesis:*

$\forall\, u_c : \Sigma^* \bullet$
$\quad u_c \in \mathit{traces}(\mathcal{B}_{Ctr_1} \interleave \mathcal{B}_{Ctr_2})$
$\quad \Rightarrow$
$\quad \exists\, u : \mathit{traces}(\mathcal{B}_{Ctr_1}) \bullet u \; \mathit{in} \; u_c \vee \exists\, u : \mathit{traces}(\mathcal{B}_{Ctr_2}) \bullet u \; \mathit{in} \; u_c$

We need to prove:

$u_c \frown \langle s\rangle \in \mathit{traces}(\mathcal{B}_{Ctr_1} \interleave \mathcal{B}_{Ctr_2})$
$\Rightarrow$
$\exists\, u : \mathit{traces}(\mathcal{B}_{Ctr_1}) \bullet u \; \mathit{in} \; (u_c \frown \langle s\rangle) \vee \exists\, u : \mathit{traces}(\mathcal{B}_{Ctr_2}) \bullet u \; \mathit{in} \; (u_c \frown \langle s\rangle)$

We start from the left-hand side of the predicate and get the right-hand side.

Proof

$$
\begin{array}{ll}
u_c \frown \langle s \rangle \in \mathit{traces}(\mathcal{B}_{Ctr_1} \interleave \mathcal{B}_{Ctr_2}) & \\
\Rightarrow & \textit{[Property of traces (Prefix-closed)]} \\
u_c \in \mathit{traces}(\mathcal{B}_{Ctr_1} \interleave \mathcal{B}_{Ctr_2}) & \\
\wedge\ u_c \frown \langle s \rangle \in \mathit{traces}(\mathcal{B}_{Ctr_1} \interleave \mathcal{B}_{Ctr_2}) & \\
\Rightarrow & \textit{[IH]} \\
\exists\, u : \mathit{traces}(\mathcal{B}_{Ctr_1}) \bullet u\ \mathit{in}\ u_c \vee \exists\, u : \mathit{traces}(\mathcal{B}_{Ctr_2}) \bullet u\ \mathit{in}\ u_c & \\
\wedge\ u_c \frown \langle s \rangle \in \mathit{traces}(\mathcal{B}_{Ctr_1} \interleave \mathcal{B}_{Ctr_2}) & \\
\Rightarrow & \textit{[Property of Subsequence and Property of traces}(P \interleave Q)\textit{]} \\
\exists\, u : \mathit{traces}(\mathcal{B}_{Ctr_1}) \bullet u\ \mathit{in}\ (u_c \frown \langle s \rangle) \vee \exists\, u : \mathit{traces}(\mathcal{B}_{Ctr_2}) \bullet u\ \mathit{in}\ (u_c \frown \langle s \rangle) &
\end{array}
$$

□

Lemma 13. *Let Ctr be a Livelock-free Component Contract, ch a communication channel in* $\mathit{Allowed}_{Bric}(Ctr)$*. Then,*

$$
\left(
\begin{array}{l}
ch \in \mathit{Allowed}_{Bric}(Ctr) \\
\Rightarrow \\
\forall\, u : \Sigma^{\omega} \mid tr^{\infty}(u, \mathcal{B}_{Ctr}) \bullet (u \setminus HC_{Bin} : \Sigma^{\omega})
\end{array}
\right)
$$

Proof

$$
\begin{array}{ll}
ch \in \mathit{Allowed}_{Bric}(Ctr) & \\
\equiv & \textit{[Definition 12 (}\mathit{Allowed}_{Bric}(Ctr)\textit{)]} \\
ch \in (\mathcal{C}_{Ctr} \setminus \{c : \mathcal{C}_{Ctr} \mid \exists\, m : \mathit{MIP}(Ctr) \bullet (\operatorname{ran}(m) \cap \mathit{evs}(\mathcal{C}_{Ctr})) \subseteq \{\mid c \mid\}\}) & \\
\Rightarrow & \textit{[Property of Sets]} \\
ch \in \mathcal{C}_{Ctr} \wedge ch \notin \{c : \mathcal{C}_{Ctr} \mid \exists\, m : \mathit{MIP}(Ctr) \bullet (\operatorname{ran}(m) \cap \mathit{evs}(\mathcal{C}_{Ctr})) \subseteq \{\mid c \mid\}\} & \\
\Rightarrow & \textit{[Predicate Calculus]} \\
\neg\ \exists\, m : \mathit{MIP}(\mathcal{B}_{Ctr}) \bullet (\operatorname{ran}(m) \cap \mathit{evs}(\mathcal{C}_{Ctr})) \subseteq \{\mid ch \mid\} & \\
\Rightarrow & \textit{[Predicate Calculus]} \\
\forall\, m : \mathit{MIP}(\mathcal{B}_{Ctr}) \bullet \neg\ ((\operatorname{ran}(m) \cap \mathit{evs}(\mathcal{C}_{Ctr})) \subseteq \{\mid ch \mid\}) & \\
\Rightarrow & \textit{[Property of Sets]} \\
\forall\, m : \mathit{MIP}(\mathcal{B}_{Ctr}) \bullet \exists\, x : \operatorname{ran}(m) \bullet x \in \mathit{evs}(\mathcal{C}_{Ctr}) \wedge x \notin \{\mid ch \mid\} & \\
\Rightarrow & \textit{[Property of Sets]} \\
\forall\, m : \mathit{MIP}(\mathcal{B}_{Ctr}) \bullet \exists\, x : \operatorname{ran}(m) \bullet x \in (\mathit{evs}(\mathcal{C}_{Ctr}) \setminus \{\mid ch \mid\}) & \\
\Rightarrow & \textit{[Property of Sets]} \\
\forall\, m : \mathit{MIP}(\mathcal{B}_{Ctr}) \bullet (\exists\, x : \operatorname{ran}(m) \bullet x \in (\mathit{evs}(\mathcal{C}_{Ctr}) \setminus \{\mid ch \mid\}) & \\
\qquad\qquad\qquad \wedge\ x \notin (\Sigma \setminus (\{\mid \mathcal{C}_{Ctr} \mid\} \setminus \{\mid ch \mid\}))) &
\end{array}
$$

$\Rightarrow$ *[Property of hiding for traces (22)]*

$\forall\, m : MIP(\mathcal{B}_{Ctr}) \bullet (\exists\, x : \mathrm{ran}(m) \bullet x \in (evs(\mathcal{C}_{Ctr}) \setminus \{\mid ch \mid\})$

$\quad \wedge\, (x \notin (\Sigma \setminus (evs(\mathcal{C}_{Ctr}) \setminus \{\mid ch \mid\})))$

$\quad \wedge\, (m \setminus (\Sigma \setminus (evs(\mathcal{C}_{Ctr}) \setminus \{\mid ch \mid\})) \neq \langle\rangle))$

$\Rightarrow$ *[Predicate Calculus]*

$\forall\, m : MIP(\mathcal{B}_{Ctr}) \bullet (m \setminus (\Sigma \setminus (evs(\mathcal{C}_{Ctr}) \setminus \{\mid ch \mid\})) \neq \langle\rangle)$

$\Rightarrow$ *[Property of Sequences]*

$\forall\, u : MIP(\mathcal{B}_{Ctr})^{\omega} \bullet (u \setminus (\Sigma \setminus (evs(\mathcal{C}_{Ctr}) \setminus \{\mid ch \mid\})) : \Sigma^{\omega})$

$\Leftrightarrow$ *[$(evs(\mathcal{C}_{Ctr}) \cap \{\mid ch_2 \mid\}) = \emptyset) \Rightarrow evs(\mathcal{C}_{Ctr}) \setminus \{\mid ch \mid\} = evs(\mathcal{C}_{Ctr}) \setminus \{\mid ch, ch_2 \mid\}$]*

$\forall\, u : MIP(\mathcal{B}_{Ctr})^{\omega} \bullet (u \setminus (\Sigma \setminus (evs(\mathcal{C}_{Ctr}) \setminus \{\mid ch, ch_2 \mid\})) : \Sigma^{\omega})$

$\Rightarrow$ *[$HC_{Bin} \subseteq (\Sigma \setminus (evs(\mathcal{C}_{Ctr}) \setminus \{\mid ch, ch_2 \mid\}))$ and Lemma 14]*

$\forall\, u : MIP(\mathcal{B}_{Ctr})^{\omega} \bullet ((u \setminus HC_{Bin}) : \Sigma^{\omega})$

$\Rightarrow$ *[Property of MIP 4.1]*

$(\forall\, u : MIP(\mathcal{B}_{Ctr})^{\omega} \bullet ((u \setminus HC_{Bin}) : \Sigma^{\omega}))$

$\wedge\, (\forall\, u : \Sigma^{\omega} \bullet (tr^{\infty}(u, \mathcal{B}_{Ctr})) \Rightarrow (u \in MIP(\mathcal{B}_{Ctr})^{\omega}))$

$\Rightarrow$ *[Instantiation of $\forall$]*

$\forall\, u : \Sigma^{\omega} \bullet (tr^{\infty}(u, \mathcal{B}_{Ctr})) \Rightarrow ((u \setminus HC_{Bin}) : \Sigma^{\omega})$

$\Rightarrow$ *[Predicate Calculus]*

$\forall\, u : \Sigma^{\omega} \mid tr^{\infty}(u, \mathcal{B}_{Ctr}) \bullet (u \setminus HC_{Bin} : \Sigma^{\omega})$

□

Lemma 14. *Let be $u : \Sigma^{\omega}$ and $Y \subseteq X$, then,*

$(u \setminus X)$ *is not finite*
$\Rightarrow$
$(u \setminus Y)$ *is not finite*

Proof by contradiction

$\neg\, ((u \setminus X)$ *is not finite* $\Rightarrow (u \setminus Y)$ *is not finite*$)$

$\Rightarrow$ *[Predicate Calculus]*

$(u \setminus X)$ *is not finite* $\wedge\, (u \setminus Y)$ *finite*

$\Rightarrow$ *[$Y \subseteq X$]*

$u \setminus (X \cup Y)$ *is not finite* $\wedge\, (u \setminus Y)$ *finite*

$\Rightarrow$ *[Property of Sets]*

$(u \setminus X)$ *is not finite* $\wedge\, (u \setminus Y)$ *is not finite* $\wedge\, (u \setminus Y)$ *finite*

$\Rightarrow$ *[Contradiction]*

false

Lemma 15. *Let Ctr be a component contract and* $Ctr \asymp|^{\langle ic\rangle}_{\langle oc\rangle}$ *an asynchronous unary composition of Ctr (Definition 4). Then, the projection of the behaviour of this composition over the set of communication channels, namely* $\mathcal{B}_{Ctr \asymp|^{(ic)}_{(oc)}} \upharpoonright \mathcal{C}_{Ctr \asymp|^{(ic)}_{(oc)}}$*, is described as follows:*

$$\mathcal{B}_{Ctr \asymp|^{(ic)}_{(oc)}} \upharpoonright \mathcal{C}_{Ctr \asymp|^{(ic)}_{(oc)}} = (\mathcal{B}_{Ctr} \parallel_{\{ic,oc\}} BUFF^{\infty}_{IO}) \setminus HC_{Unary}$$

Proof

$\mathcal{B}_{Ctr \asymp|^{\langle ic\rangle}_{\langle oc\rangle}} \upharpoonright \mathcal{C}_{Ctr \asymp|^{(ic)}_{(oc)}}$

$=$ *[Definition 4]*

$(\mathcal{B}_{Ctr} \parallel_{\{|ic,oc|\}} BUFF^{\infty}_{IO}) \upharpoonright (\mathcal{C}_{Ctr} \setminus \{|\ ic, oc\ |\})$

$=$ *[Definition 21 ($P \upharpoonright Ch$)]*

$(\mathcal{B}_{Ctr} \parallel_{\{|ic,oc|\}} BUFF^{\infty}_{IO}) \setminus (\Sigma \setminus (\mathcal{C}_{Ctr} \setminus \{|\ ic, oc\ |\}))$

$=$ *[Definition of HC_{Unary} 20]*

$(\mathcal{B}_{Ctr} \parallel_{\{|ic,oc|\}} BUFF^{\infty}_{IO}) \setminus HC_{Unary}$

□

Lemma 16. *Let* $\mathcal{B}_P$ *be an I/O Process and* $BUFF_{IO}$ *a livelock-free infinite buffer. Then, a CSP parallel composition of* $\mathcal{B}_P$ *and* $BUFF^{\infty}_{IO}$*,* $(\mathcal{B}_P \parallel_{\{X\}} BUFF^{\infty}_{IO})$*, is divergence-free. Then,*

$$divergences(\mathcal{B}_P \parallel_{\{X\}} BUFF^{\infty}_{IO}) = \emptyset$$

Proof

$divergences(\mathcal{B}_P \parallel_{\{X\}} BUFF^{\infty}_{IO})$

$=$ *[Definition 16 ($divergences(P \parallel_X Q)$)]*

$\{u \frown v \mid \exists\, s \in traces(\mathcal{B}_P), t \in traces(BUFF^{\infty}_{IO}) \bullet$
$\quad u \in (s \parallel t) \cap \Sigma^* \wedge (s \in divergences(\mathcal{B}_P)) \vee t \in divergences(BUFF^{\infty}_{IO}))\}$

$=$ *[$divergences(\mathcal{B}_P) = \emptyset$ by Definition 2]*

$\{u \frown v \mid \exists\, s \in traces(\mathcal{B}_P), t \in traces(BUFF^{\infty}_{IO}) \bullet$
$\quad u \in (s \parallel t) \cap \Sigma^* \wedge (s \in \emptyset \vee t \in divergences(BUFF^{\infty}_{IO}))\}$

$=$ *[$divergences(BUFF^{\infty}_{IO}) = \emptyset$]*

$\{u \frown v \mid \exists\, s \in traces(\mathcal{B}_P), t \in traces(BUFF^{\infty}_{IO}) \bullet u \in (s \parallel t) \cap \Sigma^* \wedge (s \in \emptyset \vee t \in \emptyset)\}$

$=$ *[Property of Sets and Predicate Calculus]*

$\{u \frown v \mid false\}$

$=$ *[Property of Sets]*

$\emptyset$

□

Lemma 17. *Let Ctr be a Livelock-free Component Contract, (ic, oc) a pair of communication channels in $Allowed_{Pair}(Ctr)$. Then,*

$$
\begin{array}{l}
(ic, oc) \in Allowed_{Pair}(Ctr) \\
\Rightarrow \\
\forall\, u : \Sigma^{\omega} \bullet tr^{\infty}(u, \mathcal{B}_{Ctr} \parallel_{\{ic,oc\}} BUFF^{\infty}_{IO}) \Rightarrow (u \setminus HC_{Unary} : \Sigma^{\omega})
\end{array}
$$

Proof

$$
\begin{array}{lr}
(ic, oc) \in Allowed_{Pair}(Ctr) & \\
\equiv & \textit{[Definition 13 (}Allowed_{Pair}(Ctr)\textit{)]} \\
(ic, oc) \in \{c_1, c_2 : \mathcal{C}_{Ctr} \mid \neg\, (\exists\, m : MIP(Ctr) \bullet (\mathrm{ran}(m) \cap evs(\mathcal{C}_{Ctr})) \subseteq \{\mid c_1, c_2 \mid\})\} & \\
\Rightarrow & \textit{[Property of Sets]} \\
\neg\, (\exists\, m : MIP(Ctr) \bullet (\mathrm{ran}(m) \cap evs(\mathcal{C}_{Ctr})) \subseteq \{\mid ic, oc \mid\}) & \\
\Rightarrow & \textit{[Predicate Calculus]} \\
\forall\, m : MIP(Ctr) \bullet \neg\, ((\mathrm{ran}(m) \cap evs(\mathcal{C}_{Ctr})) \subseteq \{\mid ic, oc \mid\}) & \\
\Rightarrow & \textit{[Property of Sets]} \\
\forall\, m : MIP(\mathcal{B}_{Ctr}) \bullet \exists\, x : \mathrm{ran}(m) \bullet x \in evs(\mathcal{C}_{Ctr}) \wedge x \notin \{\mid ic, oc \mid\} & \\
\Rightarrow & \textit{[Property of Sets]} \\
\forall\, m : MIP(\mathcal{B}_{Ctr}) \bullet \exists\, x : \mathrm{ran}(m) \bullet x \in (evs(\mathcal{C}_{Ctr}) \setminus \{\mid ic, oc \mid\}) & \\
\Rightarrow & \textit{[Property of Sets and Definition of } HC_{Unary} \textit{ 20]} \\
\forall\, m : MIP(\mathcal{B}_{Ctr}) \bullet \exists\, x : \mathrm{ran}(m) \bullet x \in (evs(\mathcal{C}_{Ctr}) \setminus \{\mid ic, oc \mid\}) \wedge x \notin HC_{Unary} & \\
\Rightarrow & \textit{[Property of Hiding for Traces (22)]} \\
\forall\, m : MIP(\mathcal{B}_{Ctr}) \bullet \exists\, x : \mathrm{ran}(m) \bullet x \in (evs(\mathcal{C}_{Ctr}) \setminus \{\mid ic, oc \mid\}) \wedge x \notin HC_{Unary} & \\
\qquad\qquad\qquad\qquad\qquad\qquad \wedge\ ((m \setminus HC_{Unary}) \neq \langle\rangle) & \\
\Rightarrow & \textit{[Predicate Calculus]} \\
\forall\, m : MIP(\mathcal{B}_{Ctr}) \bullet ((m \setminus HC_{Unary}) \neq \langle\rangle) & \\
\Rightarrow & \textit{[Property of Sequences and Property of MIP‘ 4.1]} \\
\forall\, u : MIP(\mathcal{B}_{Ctr})^{\omega} \bullet ((u \setminus HC_{Unary}) : \Sigma^{\omega}) & \\
\Rightarrow & \textit{[Property of MIP 4.1]} \\
\forall\, u : MIP(\mathcal{B}_{Ctr})^{\omega} \bullet ((u \setminus HC_{Unary}) : \Sigma^{\omega}) & \\
\wedge\ \forall\, u : \Sigma^{\omega} \bullet tr^{\infty}(u, \mathcal{B}_{Ctr}) \Rightarrow u \in MIP(\mathcal{B}_{Ctr})^{\omega} & \\
\Rightarrow & \textit{[Instantiation of } \forall \textit{]} \\
\forall\, u : \Sigma^{\omega} \bullet tr^{\infty}(u, \mathcal{B}_{Ctr}) \Rightarrow (u \setminus HC_{Unary} : \Sigma^{\omega}) & \\
\Rightarrow & \textit{[Lemma 9]} \\
\forall\, u : \Sigma^{\omega} \bullet tr^{\infty}(u, \mathcal{B}_{Ctr} \parallel_{\{ic,oc\}} BUFF^{\infty}_{IO}) \Rightarrow (u \setminus HC_{Unary} : \Sigma^{\omega}) &
\end{array}
$$

□

Lemma 18. *Let Ctr_1 and Ctr_2 be two livelock-free component contracts, ic a communication channel in $Allowed(Ctr_1)$, and oc a communication channel in $\mathcal{C}_{Ctr_2}$. Then;*

$$\left(\begin{array}{l} ic \in Allowed(Ctr_1) \\ \Rightarrow \\ divergences(\mathcal{B}_{Ctr_1[ic \leftrightarrow oc]Ctr_2} \upharpoonright \mathcal{C}_{Ctr_1[ic \leftrightarrow oc]Ctr_2}) = \emptyset \end{array}\right)$$

We assume that $ic \in Allowed(Ctr_1)$ and prove that:

$$divergences(\mathcal{B}_{Ctr_1[ic \leftrightarrow oc]Ctr_2} \upharpoonright \mathcal{C}_{Ctr_1[ic \leftrightarrow oc]Ctr_2}) = \emptyset$$

Proof

$divergences(\mathcal{B}_{Ctr_1[ic \leftrightarrow oc]Ctr_2} \upharpoonright \mathcal{C}_{Ctr_1[ic \leftrightarrow oc]Ctr_2})$

$=$ *[Definition 6]*

$divergences(\mathcal{B}_{Ctr_1\,\langle ic\rangle \asymp \langle oc\rangle\, Ctr_2} \upharpoonright \mathcal{C}_{Ctr_1\,\langle ic\rangle \asymp \langle oc\rangle\, Ctr_2})$

$=$ *[Lemma 3]*

$divergences(((\mathcal{B}_{Ctr_1} \interleave \mathcal{B}_{Ctr_2}) \parallel_{\{ic,oc\}} BUFF^{*}_{IO}) \setminus HC_{Bin})$

$=$ *[Lemma 6, Lemma 4 and Lemma 5]*

$\{(u \setminus HC_{Bin}) \frown t \mid u : \Sigma^{\omega} \wedge \neg\,(u \setminus HC_{Bin}) : \Sigma^{\omega}$
$\qquad \wedge \forall\, s < u \bullet s \in traces((\mathcal{B}_{Ctr_1} \interleave \mathcal{B}_{Ctr_2}) \parallel_{\{ic,oc\}} BUFF^{*}_{IO})\}$

$=$ *[Definition of $tr^{\infty}(u, (\mathcal{B}_{Ctr_1} \interleave \mathcal{B}_{Ctr_2}) \parallel_{\{ic,oc\}} BUFF^{*}_{IO})$ (18)]*

$\{(u \setminus HC_{Bin}) \frown t \mid u : \Sigma^{\omega} \wedge \neg\,(u \setminus HC_{Bin}) : \Sigma^{\omega}$
$\qquad \wedge\, tr^{\infty}(u, (\mathcal{B}_{Ctr_1} \interleave \mathcal{B}_{Ctr_2}) \parallel_{\{ic,oc\}} BUFF^{*}_{IO})\}$

$=$ *[Lemma 19]*

$\{(u \setminus HC_{Bin}) \frown t \mid u : \Sigma^{\omega} \wedge \neg\,(u \setminus HC_{Bin}) : \Sigma^{\omega}$
$\qquad \wedge\, tr^{\infty}(u, (\mathcal{B}_{Ctr_1} \interleave \mathcal{B}_{Ctr_2}) \parallel_{\{ic,oc\}} BUFF^{*}_{IO})$
$\qquad \wedge\, u \setminus HC_{Bin} : \Sigma^{\omega}\}$

$=$ *[Property of Sets and Predicate Calculus]*

$\{(u \setminus HC_{Bin}) \frown t \mid false\}$

$=$ *[Property of Sets]*

$\emptyset$

$\square$

Lemma 19. *Let Ctr_1 and Ctr_2 be two livelock-free contracts, ic a communication channel in Allowed(Ctr_1), and oc a communication channel in $\mathcal{C}_{Ctr_2}$, such that $\mathcal{C}_{Ctr_1} \cap \mathcal{C}_{Ctr_2} = \emptyset$. Then,*

$$\forall\, u_c : \Sigma^\omega \bullet tr^\infty(u_c, (\mathcal{B}_{Ctr_1} \interleave \mathcal{B}_{Ctr_2}) \parallel_{\{|ic,oc|\}} BUFF^*_{IO}) \Rightarrow (u_c \setminus HC_{Bin}) : \Sigma^\omega$$

provided $ic \in Allowed(Ctr_1)$

Proof

$$
\begin{array}{ll}
tr^\infty(u_c, (\mathcal{B}_{Ctr_1} \interleave \mathcal{B}_{Ctr_2}) \parallel_{\{|ic,oc|\}} BUFF^*_{IO}) & \\
\Rightarrow & \textit{[Infinite traces of } ((\mathcal{B}_{Ctr_1} \interleave \mathcal{B}_{Ctr_2}) \parallel_{\{|ic,oc|\}} BUFF_{IO})\textit{]} \\
\exists\, u_1 : \Sigma^\omega \bullet tr^\infty(u_1, \mathcal{B}_{Ctr_1}) \wedge u_1\ in^\infty\ u_c & \\
\wedge\ \exists\, u_2 : \Sigma^\omega \bullet tr^\infty(u_2, \mathcal{B}_{Ctr_2}) \wedge u_2\ in^\infty\ u_c & \\
\Rightarrow & \textit{[Predicate Calculus]} \\
\exists\, u_1, u_2 : \Sigma^\omega \bullet ((tr^\infty(u_1, \mathcal{B}_{Ctr_1}) \wedge u_1\ in^\infty\ u_c) & \\
\qquad \wedge\ (tr^\infty(u_2, \mathcal{B}_{Ctr_2}) \wedge u_2\ in^\infty\ u_c)) & \\
\Rightarrow & \textit{[Predicate Calculus]} \\
\exists\, u_1, u_2 : \Sigma^\omega \bullet ((tr^\infty(u_1, \mathcal{B}_{Ctr_1}) \wedge tr^\infty(u_2, \mathcal{B}_{Ctr_2})) & \\
\qquad \wedge\ (u_1\ in^\infty\ u_c \wedge u_2\ in^\infty\ u_c)) & \\
\Rightarrow & \textit{[Predicate Calculus]} \\
\exists\, u_1, u_2 : \Sigma^\omega \mid ((tr^\infty(u_1, \mathcal{B}_{Ctr_1}) \wedge tr^\infty(u_2, \mathcal{B}_{Ctr_2})) & \\
\qquad\qquad \bullet (u_1\ in^\infty\ u_c \wedge u_2\ in^\infty\ u_c)) & \\
\equiv & \textit{[Property of Hiding for Sequences]} \\
\exists\, u_1, u_2 : \Sigma^\omega \mid ((tr^\infty(u_1, \mathcal{B}_{Ctr_1}) \wedge tr^\infty(u_2, \mathcal{B}_{Ctr_2})) & \\
\qquad\qquad \bullet (u_1 \setminus HC_{Bin}\ in^\infty\ u_c \setminus HC_{Bin} \wedge u_2 \setminus HC_{Bin}\ in^\infty\ u_c \setminus HC_{Bin})) & \\
\Rightarrow & \textit{[Lemma 13 and proviso]} \\
\exists\, u_1, u_2 : \Sigma^\omega \mid ((tr^\infty(u_1, \mathcal{B}_{Ctr_1}) \wedge tr^\infty(u_2, \mathcal{B}_{Ctr_2})) & \\
\qquad\qquad \bullet (u_1 \setminus HC_{Bin}\ in^\infty\ u_c \setminus HC_{Bin} \wedge u_1 \setminus HC_{Bin} : \Sigma^\omega & \\
\qquad\qquad\quad \wedge\ u_2 \setminus HC_{Bin}\ in^\infty\ u_c \setminus HC_{Bin})) & \\
\Rightarrow & \textit{[Property of Infinite Sequences } (s : \Sigma^\omega \wedge s\ in^\infty\ t \Rightarrow t : \Sigma^\omega)\textit{]} \\
\exists\, u_1, u_2 : \Sigma^\omega \mid ((tr^\infty(u_1, \mathcal{B}_{Ctr_1}) \wedge tr^\infty(u_2, \mathcal{B}_{Ctr_2})) & \\
\qquad\qquad \bullet (u_c \setminus HC_{Bin} : \Sigma^\omega \wedge u_2 \setminus HC_{Bin}\ in^\infty\ u_c \setminus HC_{Bin})) & \\
\Rightarrow & \textit{[Predicate Calculus]} \\
\exists\, u_1, u_2 : \Sigma^\omega \mid ((tr^\infty(u_1, \mathcal{B}_{Ctr_1}) \wedge tr^\infty(u_2, \mathcal{B}_{Ctr_2})) & \\
\qquad\qquad \bullet (u_c \setminus HC_{Bin} : \Sigma^\omega) & \\
\Rightarrow & \textit{[Predicate Calculus]}
\end{array}
$$

$u_c \setminus HC_{Bin} : \Sigma^\omega$

$\square$

Lemma 20. *Let Ctr be a livelock-free component contract, ic and oc two communication channels in $\mathcal{C}_{Ctr}$, such that in $\in$ Allowed(Ctr), and $\{ic, oc\}$ are decoupled in Ctr. Then;*

$$\left(\begin{array}{l} ic \in Allowed(Ctr) \\ \Rightarrow \\ divergences(\mathcal{B}_{Ctr[oc \hookrightarrow ic]^*} \upharpoonright \mathcal{C}_{Ctr[oc \hookrightarrow ic]^*}) = \emptyset \end{array}\right)$$

We assume that $ic \in Allowed(Ctr)$ and prove that:

$$divergences(\mathcal{B}_{Ctr[oc \hookrightarrow ic]^*} \upharpoonright \mathcal{C}_{Ctr[oc \hookrightarrow ic]^*}) = \emptyset$$

Proof

$divergences(\mathcal{B}_{Ctr[oc \hookrightarrow ic]^*} \upharpoonright \mathcal{C}_{Ctr[oc \hookrightarrow ic]^*})$

$=$ *[Definition 7]*

$divergences(\mathcal{B}_{Ctr \preccurlyeq|^{\langle ic \rangle}_{\langle oc \rangle}} \upharpoonright \mathcal{C}_{Ctr \preccurlyeq|^{\langle ic \rangle}_{\langle oc \rangle}})$

$=$ *[Lemma 15]*

$divergences((\mathcal{B}_{Ctr} \parallel_{\{ic, oc\}} BUFF^*_{IO}) \setminus HC_{Unary})$

$=$ *[Lemma 4, Lemma 16, and Definition of $tr^\infty(u, (\mathcal{B}_{Ctr} \parallel_{\{ic, oc\}} BUFF^*_{IO}))$ (18)]*

$\{(u \setminus HC_{Unary}) ^\frown t \mid u : \Sigma^\omega \wedge \neg\, (u \setminus HC_{Unary}) : \Sigma^\omega$
$\wedge\ tr^\infty(u, (\mathcal{B}_{Ctr} \parallel_{\{ic, oc\}} BUFF^*_{IO}))\}$

$=$ *[$\{ic, oc\}$ are decoupled in Ctr, and Lemma 13]*

$\{(u \setminus HC_{Unary}) ^\frown t \mid u : \Sigma^\omega \wedge \neg\, (u \setminus HC_{Unary}) : \Sigma^\omega$
$\wedge\ tr^\infty(u, (\mathcal{B}_{Ctr} \parallel_{\{ic, oc\}} BUFF^*_{IO})) \wedge u \setminus HC_{Unary} : \Sigma^\omega\}$

$=$ *[Property of Sets and Predicate Calculus]*

$\{(u \setminus HC_{Unary}) ^\frown t \mid false\}$

$=$ *[Property of Sets]*

$\emptyset$

$\square$

The following process presents the CSP specification for finite buffers.

$$BUFF^{*}_{IO}(n) = BF1(n) \mid\mid\mid BF2(n)$$

```
$BF1(n) =$
    let $B(\langle\rangle) = in?x \rightarrow B(\langle x \rangle)$
        $B(s) = (out!head(s) \rightarrow B(tail(s)))$
                $\Box\ (\#s < n\ \&\ in?x \rightarrow B(s \frown \langle x \rangle))$
    within $B(\langle\rangle)$
```

```
$BF2(n) =$
    let $B(\langle\rangle) = out?x \rightarrow B(\langle x \rangle)$
        $B(s) = (in!head(s) \rightarrow B(tail(s)))$
                $\Box\ (\#s < n\ \&\ out?x \rightarrow B(s \frown \langle x \rangle))$
    within $B(\langle\rangle)$
```

The following process presents the CSP specification for infinite buffers.

$$BUFF^{\omega}_{IO}(n) = BF1(n) \mid\mid\mid BF2(n)$$

```
$BF1(n) =$
    let $B(\langle\rangle) = in?x \rightarrow B(\langle x \rangle)$
        $B(s) = (out!head(s) \rightarrow B(tail(s)))$
                $\Box\ in?x \rightarrow B(s \frown \langle x \rangle)$
    within $B(\langle\rangle)$
```

```
$BF2(n) =$
    let $B(\langle\rangle) = out?x \rightarrow B(\langle x \rangle)$
        $B(s) = (in!head(s) \rightarrow B(tail(s)))$
                $\Box\ out?x \rightarrow B(s \frown \langle x \rangle)$
    within $B(\langle\rangle)$
```

8 Appendix B

In this section we present some auxiliary definitions used in our proofs.

Definition 16 (Divergences semantics of processes). *Let P and Q be two CSP processes.*

$$
\begin{array}{lcl}
divergences(SKIP) & = & \emptyset \\
divergences(STOP) & = & \emptyset \\
divergences(a \rightarrow P) & = & \{\langle a \rangle \frown s \mid s \in divergences(P)\} \\
divergences(P \,\square\, Q) & = & divergences(P) \cup divergences(Q) \\
divergences(P \sqcap Q) & = & divergences(P) \cup divergences(Q) \\
divergences(P[[R]]R) & = & \{s \frown u \mid \exists\, t \in divergences(P) \bullet t\, R^*\, r\} \\
divergences(P \parallel_X Q) & = & \{u \frown v \mid \exists\, s \in traces_\perp(P), t \in traces_\perp(Q) \bullet \\
& & u \in (s \parallel_X t) \cap \Sigma^* \\
& & \wedge\, (s \in divergences(P) \vee t \in divergences(Q))\} \\
divergences(P \setminus X) & = & \{(s \setminus X) \frown t \mid s \in divergences(P)\} \\
& & \cup \{(u \setminus X) \frown t \mid u \in \Sigma^\omega \wedge \neg\, (u \setminus X) : \Sigma^\omega \\
& & \wedge\, \forall\, s < u \bullet s \in traces_\perp(P)\}
\end{array}
$$

where Σ^ω is the set of infinite traces.

Definition 17. *Let P be a CSP processes.*

$$traces_\perp(P) = traces(P) \cup divergences(P)$$

Definition 18. *Let P be a CSP processes.*
$tr^\infty(u, P) = (\forall\, s < u \bullet s \in traces(P))$

Definition 19. $HC_{Bin} = (\Sigma \setminus \mathcal{C}_{Ctr_1} \cup \mathcal{C}_{Ctr_2}) \cup \{ic, oc\}$

Definition 20. $HC_{Unary} = (\Sigma \setminus \mathcal{C}_{Ctr}) \cup \{ic, oc\}$

Definition 21. *Let P be a CSP processes, ans X a set of events.*
$P \upharpoonright X = P \setminus (\Sigma \setminus X)$

Definition 22. *Let $s \in \Sigma^*$ be a trace and $X \subseteq \Sigma$.*

$$s \setminus X = s \upharpoonright (\Sigma \setminus X)$$

9 Appendix C

In this section we present some refinement laws used in our proofs.

Law 1 (Hiding join)
$A \setminus (cs_1 \cup cs_2) = (A \setminus cs_1) \setminus \ cs_2$

Law 2 (Hiding/Interlerave distribution)
$(A_1 \interleave A_2) \setminus cs = (A_1 \setminus cs) \interleave (A_2 \setminus cs)$

Law 3 (Interleave/Skip)
$\interleave_{c \in \emptyset} A = SKIP$

Law 4 (Interleave unit)
$A \interleave Skip = A$

Law 5 (Parallel/Interleave)
$A_1 \parallel_{\{\}} A_2 = A_1 \interleave A_2$