UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
DEPARTAMENTO DE INFORMÁTICA E MATEMÁTICA APLICADA (DIMAP)
PROGRAMA DE PÓS-GRADUAĆÃO EM SISTEMAS E COMPUTAĆÃO

Proof of laws from the operational semantics of *Circus*

Samuel Lincoln Magalhães Barrocas

Orientador: Prof. Dr. Marcel Vinícius Medeiros Oliveira

Technical Report

Natal, RN, July 2017

Contents

Appendix A

Circus Denotational Semantics

On this annex, we will show some used Denotational Semantics laws of *Circus*. All these laws were extracted from [6].

Definition A.1. ***Variable Block Denotational Definition**:*
var $x : T \bullet A = \exists\, x, x' : T \,.\, A$

Proof The law was created on document [6].. □

Definition A.2. ***Prefixing Denotational Definition***
$l \rightarrow Skip = R\,(true \vdash doC\,(l, C) \wedge vars' = vars)$

Where

- *l is a communication*
- *doC is defined on B.32*
- *v is the set of variables of the state space (not including either ok, wait, ref or tr)*

The definition was created on [6].

Definition A.3. ***Skip Denotational Definition**:*
$Skip = R(true \vdash tr' = tr \wedge \neg\, wait' \wedge v' = v)$

The definition was created on [6].

Definition A.4. ***Assignment Denotational Definition**:*
$x := e = R\,(true \vdash x' = e \wedge u' = u \wedge \neg\, wait')$

Where u = v - {x}. The definition was created on [6].

Definition A.5. ***If-Guarded Command Denotational Definition**:*
if $[\!] \; i \bullet g_i \rightarrow A_i$ **fi** $= R\,((\bigvee i \bullet g_i) \wedge (\bigwedge i \bullet g_i \Rightarrow \neg\ A_i{}^{f}_{f}) \vdash \bigvee i \bullet (g_i \wedge A_i{}^{t}_{f}))$

The definition lies on [6].

Definition A.6. ***Specification Statement Denotational Definition**:*
$w : [pre, post] = R\,(pre \vdash post \wedge \neg\ wait' \wedge tr' = tr \wedge u' = u)$

The definition lies on [6].

Appendix B

Proofs of rules of the Operational Semantics of *Circus*

On this appendix we will show the proofs of Soundness of the rules of the Operational Semantics of *Circus* we developed for this thesis. The rules were proved sound with respect to the Denotational Semantics of *Circus*, that is the theory of *Circus* in the Unifying Theories of Programming.

Each proof can be of a lemma, a theorem, a law or a rule from the Operational Semantics of *Circus*. Each proof has a sequence of sub-goals, that are intermediate expressions that compose the proof, and tactics, that transform a sub-goal on another sub-goal. The format of each proof is shown as follows:

Sub-goal 1
Op [Tactic]
...
Op [Tactic]
Sub-goal n

Each tactic is highlighted in **bold**, and each operator **Op** can be either "=", "$\Rightarrow$" or "$\sqsubseteq$". When the tactic uses either a theorem, a lemma, a rule or a definition, it will indicate what theorem, lemma, rule or definition it is and having an hyperlink to it.

B.1 Proving the Soundness of a rule

To prove that a rule of the Operational Semantics of *Circus* is sound with respect to the Denotational Semantics of *Circus* (the theory of *Circus* on the Unifying Theories of Programming) means that the transition from the rule is a predicate that is **true**. We will

exemplify a proof of Soundness for the rule of the Guard construct. The rule for Guard construct was created by [3] and proved correct on this PhD thesis (see table B.1). The rule is shown as follows:

$$\frac{c \qquad (s\ ;\ g)}{(c \mid s \models g\ \&\ A) \xrightarrow{\tau} (c \wedge (s\ ;\ g) \mid s \models A)}$$

Thus, what we want is to show that the silent transition

$$(c \mid s \models g\ \&\ A) \xrightarrow{\tau} (c \wedge (s\ ;\ g) \mid s \models A)$$

is a predicate that is **true**. The factors that above the line, *c* and *s*; *g*, are provisos of the proof: a proviso is an expression that is assumed **true** for the proof. So, the rule assumes that *c* is **true** and that *s*; *g* is **true**.

Before proving the expression above, we will create a theorem and prove it: it is called Refinement Equal Sides:

Theorem - Refinement Equal Sides:

EXPR $\sqsubseteq$ EXPR

Proof:

EXPR $\sqsubseteq$ EXPR

(The expression we want to prove is the one above. A refinement expression
E1 $\sqsubseteq$ E2 equals [E2 $\Rightarrow$ E1], where the brackets mean universal quantification on the free-variables of the expression:)
= [EXPR $\Rightarrow$ EXPR]

(Now we apply a Predicate Calculus: an implication expression consists on a disjunction ("or" operator) between the negation of the left side and the right side:)
= [$\neg$ EXPR $\vee$ EXPR]
= [true]
= true

We start our proof with the first sub-goal, which is the expression itself:

Proof:

$(c \mid s \models g \;\&\; A) \xrightarrow{\tau} (c \wedge (s \,;\, g) \mid s \models A)$

(As we saw from the definitions of Silent and Labelled Transition (see **??** *and* **??***), a transition on the Operational Semantics is, in fact, a predicate expression on the UTP. So, the first tactic that we will apply on the proof is to transform the operational representation of the transition into its denotational predicate expression, resulting on the predicate shown below)*
$= \forall\, w \,.\, (c \wedge c \wedge (s \,;\, g)) \Rightarrow ((\text{Lift}\ (s) \,;\, g \;\&\; A) \sqsubseteq (\text{Lift}\ (s) \,;\, A))$

(Our second tactic deals with the presence of variable w being universally quantified. When we prove that a given predicate is true, we prove that it is true for all possible values of its variables. So, during a proof, the whole expression is quantified, including w. So we can abstract the occurrence of w:)
$= (c \wedge c \wedge (s \,;\, g)) \Rightarrow ((\text{Lift}\ (s) \,;\, g \;\&\; A) \sqsubseteq (\text{Lift}\ (s) \,;\, A))$

(We have, as an assumption, that c is true. So, our tactic now is to replace all occurrences of c by true)
$= (\text{true} \wedge \text{true} \wedge (s \,;\, g)) \Rightarrow ((\text{Lift}\ (s) \,;\, g \;\&\; A) \sqsubseteq (\text{Lift}\ (s) \,;\, A))$

(Our tactic now is to apply a predicate calculus such that true $\wedge$ P equals P)
$= (\text{true} \wedge (s \,;\, g)) \Rightarrow ((\text{Lift}\ (s) \,;\, g \;\&\; A) \sqsubseteq (\text{Lift}\ (s) \,;\, A))$

(We apply, again, the same tactic: true $\wedge$ P equals P)
$= ((s \,;\, g)) \Rightarrow ((\text{Lift}\ (s) \,;\, g \;\&\; A) \sqsubseteq (\text{Lift}\ (s) \,;\, A))$

(We now divide the proof in two parts: one in which we prove the above expression is true for g being ***true****, and the other in which we prove that the above expression is* ***true*** *for g being false. If we prove that, for both cases, the above expression is true, then it is true for any value of g)*

Part 1 - for g = true:

(s ; g) ⇒ ((Lift (s) ; g & A) ⊑ (Lift (s) ; A)))

(We simply replace g by true)
= (s ; true) ⇒ ((Lift (s) ; true & A ⊑ (Lift (s) ; A))

(This tactic now transforms expression true & A into A using a theorem called True Guard, that says that g & A = A. This theorem lies on [6])
= (s ; true) ⇒ ((Lift (s) ; A) ⊑ (Lift (s) ; A)

= (s ; true) ⇒ ((Lift (s) ; A) ⊑ (Lift (s) ; A))

(Now we apply the theorem we have proved correct: Refinement Equal Sides)
= (s ; true) ⇒ true

(We apply now Predicate Calculus: when the consequent of an implication is true, then the whole expression is true:)
= **true**

Part 2 - for g = false:

(s ; g) ⇒ ((Lift (s) ; g & A) ⊑ (Lift (s) ; A)))

(We simply replace g by false)
= (s ; false) ⇒ ((Lift (s) ; false & A) ⊑ (Lift (s) ; A))

(We remove w from the expression using a similar strategy from Part 1:)
= (s ; false) ⇒ ((Lift (s) ; false & A) ⊑ (Lift (s) ; A))

(There is a theorem, on [4], that says that s; false equals false. So we make the replacement:)
= false ⇒ ((Lift (s) ; false & A) ⊑ (Lift (s) ; A))

(By Predicate Calculus, if we have false on the antecedent of an implication, then the whole expression is true)
= true

The proof for rule of Guard is shown at 11.

Beyond the proofs of soundness that we have made for the rules of *Circus* (some of which created by ourselves), we also encompass, on this document, previous results on the soundness of the Operational Semantics of Woodcock's Operational Semantics. We will show a summary of rules.

Rule (CML/Circus Actions)	Created by	Proved by
Assignment	(Woodcock et al 2013)	(Woodcock et al 2013)
Input (Prefixing)	(Woodcock et al 2013)	(Woodcock et al 2013)
Output (Prefixing)	(Woodcock et al 2013)	(Woodcock et al 2013)
Variable Block Begin	(Woodcock et al 2013)	Samuel Barrocas
Variable Block Visible	(Woodcock et al 2013)	Samuel Barrocas
Variable Block End	(Woodcock et al 2013)	Samuel Barrocas
Sequence Progress	(Woodcock et al 2013)	Samuel Barrocas
Sequence End	(Woodcock et al 2013)	Samuel Barrocas
Internal Choice Left	(Woodcock et al 2013)	(Woodcock et al 2013)
Internal Choice Right	(Woodcock et al 2013)	(Woodcock et al 2013)
Guard	(Woodcock et al 2013)	Samuel Barrocas
External Choice Begin	(Woodcock et al 2013)	(Woodcock et al 2013)
External Choice End	(Woodcock et al 2013)	(Woodcock et al 2013)
External Choice Skip	(Woodcock et al 2013)	(Woodcock et al 2013)
Ext. Choice Silent Left	(Woodcock et al 2013)	Samuel Barrocas
Ext. Choice Silent Right	(Woodcock et al 2013)	Samuel Barrocas
Parallel Begin	(Woodcock et al 2013)	(Woodcock et al 2013)
[1]Parallel Ind. Left (PIL)	(Woodcock et al 2013)	Samuel Barrocas
[2]Parallel Indep. Right	(Woodcock et al 2013)	Samuel Barrocas
Parallel Synchronised	(Woodcock et al 2013)	Samuel Barrocas
Parallel End	(Woodcock et al 2013)	(Woodcock et al 2013)
Hiding Internal	(Woodcock et al 2013)	Samuel Barrocas
Hiding Visible	(Woodcock et al 2013)	Samuel Barrocas
Hiding Skip	Samuel Barrocas	Samuel Barrocas
Recursion	(Woodcock et al 2013)	Samuel Barrocas
If-Guarded-Command	Samuel Barrocas	Samuel Barrocas
Call Action No Params	Samuel Barrocas	Samuel Barrocas
Call Action W/Params	Samuel Barrocas	Samuel Barrocas
Rename Action	Samuel Barrocas	Samuel Barrocas
Alphabetised Par. Act.	Samuel Barrocas	Not proved (future work)
Parameter Action Call	Samuel Barrocas	Samuel Barrocas
Iterated Actions	Samuel Barrocas	Samuel Barrocas
Parameter Action	Samuel Barrocas	Samuel Barrocas
Specification Statement	Samuel Barrocas	Samuel Barrocas
Schema Action	Samuel Barrocas[3]	Samuel Barrocas

Circus process	Created by	Proved by
Basic Process Begin	Samuel Barrocas[4]	Samuel Barrocas
Basic Process Reduction	Samuel Barrocas	Samuel Barrocas
Hiding Advance	Samuel Barrocas	Samuel Barrocas
Hiding Basic Process	Samuel Barrocas	Samuel Barrocas
Rename Advance	Samuel Barrocas	Samuel Barrocas
Rename Basic Process	Samuel Barrocas	Samuel Barrocas
Compound Process Left	Samuel Barrocas	Samuel Barrocas
Compound Process Right	Samuel Barrocas	Samuel Barrocas
Call process with normal parameters	Samuel Barrocas	Samuel Barrocas
Call process with indexed parameters	Samuel Barrocas	Samuel Barrocas
Call process with generic parameters	Samuel Barrocas	Samuel Barrocas
Iterated Processes	Samuel Barrocas	Samuel Barrocas

[1] and [2]: in the case of Parallel Independent Left and Parallel Independent Right, (Woodcock et al 2013) created a section to prove the soundness of these rules, but did not prove it; [3]: this rule was adapted from [1] and proved correct on this PhD thesis;

B.2 Auxiliary Definitions, Lemmas, Theorems and Laws

This appendix shows all definitions, lemmas, theorems and laws used on (or created by) this document. Those marked with a * consist on definitions that were created or quoted by other authors (quoted after each definition). Those without * were created on this document. We will begin showing Auxiliary Definitions.

B.2.1 Auxiliary Definitions

Definition B.1. ***getAssignments*** *is an auxiliary function (created for this document) that calculates the sequence of assignments of a node whose program text is a process.*
When the parameter of getAssignments is a Circus action Act, it returns the assignments s of the node whose program text is Act:

$$\frac{(c_1 \mid s_1 \models A_1) \xrightarrow{a} (c_2 \mid s_2 \models A_2)}{getAssignments\ (A_1) = s_1 \land getAssignments\ (A_2) = s_2}$$

For Example:

$(\textbf{true} \mid \{\,\} \models a \rightarrow STOP) \xrightarrow{a} (\textbf{true} \mid \{\,\} \models STOP)$
$getAssignments\ (a \rightarrow STOP) = \{\,\}$

When the parameter is a Circus process, getAssignments has the following definition:

- *getAssignments (***begin state** *ST = [decl | inv] • Act* **end***) = getAssignments (Act)*
- *getAssignments (Call) = getAssignments (Content(Call))*
- *getAssignments* $(P_1$ ***bop*** $P_2)$ *= getAssignments* $(P_1) \land$ *getAssignments* (P_2)*, where* ***bop*** *is a binary operator (either an external choice, internal choice, interleaving, parallelism and etc)*
- *getAssignments* $(P \setminus CS)$ *= getAssignments (P)*
- *getAssignments (P [RenamingPairs]) = getAssignments (P)*
- *getAssignments (ITOP Decl • P) = getAssignments (IteratedExpansion (P, Decl, ITOP))*
- *getAssignments (P [N+]) = getAssignments (GenericContent (P, [N+]))*

Definition B.2. ***pre*** *is an unary function that calculates the precondition of a given Z Schema. This calculation is performed by guaranteeing that there is a future state and by hiding the output variables of the schema (those that have a "!" after their names on the definition of the schema):*

pre *Schema* = $\exists$*State' . Schema* $\setminus$ *Outputs*

This definition lies on [9].

Definition B.3. ***IteratedExpansion*** *is a function that unfolds an iterated operator applied to an action or to a process. It has three parameters: the first parameter is the process itself, the second is the declaration of the variables used to iterate the process (or the action), and the third is a flag indicating what is the operator applied to the iteration. The type of the iterating variables must be finite, or infinite with range previously specified. For Example:*

IteratedExpansion (a.x $\rightarrow$ *Skip, x :* $\{0, 1, 2, 3\}$*, EXTCHOICE) =*
a.0 $\rightarrow$ *Skip* $\Box$ *a.1* $\rightarrow$ *Skip* $\Box$ *a.2* $\rightarrow$ *Skip* $\Box$ *a.3* $\rightarrow$ *Skip*

Definition B.4. ***Body*** *is a function that unfolds the body of a call action. For Example:*

process $P \mathrel{\widehat{=}}$ **begin**
$A \mathrel{\widehat{=}} a \rightarrow Skip$
$\bullet$ A
end

Body (A) = a $\rightarrow$ *Skip.*

Body *can also be used for processes. For Example:*

Body (P) =

begin
$A \mathrel{\widehat{=}} a \rightarrow Skip$
$\bullet$ A
end

Definition B.5. ***IndexedContent*** *works as* ***Body****, but is applied to processes with indexed parameters.*
process $P \mathrel{\widehat{=}}$ **begin** • *request?x* $\rightarrow$ *response!x* $\rightarrow$ *Skip* **end**
process $Q \mathrel{\widehat{=}} x : \{0, 1\} \odot P$
IndexedContent (Q, [0]) = $Q \lfloor\!\lfloor 0 \rfloor\!\rfloor$
= **begin** • *request_0.0?x* $\rightarrow$ *response_0.0!0* $\rightarrow$ *Skip* **end**

Definition B.6. ***ParamContent*** *works as* ***Body****, but is applied to processes with normal parameters. For Example:*

process $P \mathrel{\widehat{=}} x : \mathbb{N};\ y : \mathbb{N}$ • **begin**
• *a.x.y* $\rightarrow$ *Skip*
end

ParamContent (P, [1, 2]) = **begin** • *a.1.2* $\rightarrow$ *Skip* **end**

Definition B.7. ***GenericContent*** *works as* ***Content****, but is applied to processes with generic parameters. GenericContent (P, [N+]) takes process P and a list of generic types [N+] as parameters. For Example:*

channel *[C] singleevent : C*
process *P [T]* $\mathrel{\widehat{=}}$ *begin* • *singleevent [T]?x* $\rightarrow$ *Skip* **end**
process *Inst* $\mathrel{\widehat{=}}$ *P [* $\mathbb{N}$ *]*

On the specification above,

GenericContent (P, [$\mathbb{N}$ *])* =
begin • *singleevent [* $\mathbb{N}$ *]?x* $\rightarrow$ *Skip* **end**

Definition B.8. *A = Body (A), provided that A is a call.*

Definition B.8 says that any call action A equals its body. Given:

process $P \mathrel{\widehat=}$ **begin**
$A \mathrel{\widehat=} a \rightarrow Skip$
$\bullet\ A$
end

The call action A equals a → Skip, which is Body (A) (def. B.4). Thus, A = Body (A).

Definition B.9. *A (N+) = ParamContent (A, [N+]), provided that A is a parameterised call.*

Definition B.9 says that any parameterised call action A equals its body having the parameters replaced by their call expressions. Given:

process $P \mathrel{\widehat=}$ **begin**
$A \mathrel{\widehat=} x : \mathbb{N} \bullet a.x \rightarrow Skip$
$\bullet\ A\,(0)$
end

The call action A (0) equals a.0 → Skip, which is ParamContent (A, [0]) (def. B.6). Thus, A (0) = ParamContent (A, [0]).

Definition B.10. *A ⌊ N+ ⌋ = IndexedContent (A, [N+]), provided that A is an indexed call. Given:*

process $P \mathrel{\widehat=}$ **begin**
$A \mathrel{\widehat=} x : \mathbb{N} \bullet a.x \rightarrow Skip$
$\bullet\ A\lfloor 0 \rfloor$
end

The indexed call action A ⌊ 0 ⌋ equals a_0.0 → Skip, which is IndexedContent (A, [0]) (def. B.5). Thus, A ⌊ 0 ⌋ = IndexedContent (A, [0]).

Definition B.11. *P [N+] = GenericContent (P, [N+]), provided that P is a call. Given the example on B.7,*

$P\ [\ \mathbb{N}\ ] =$ **begin** $\bullet$ *singleevent* $[\ \mathbb{N}\ ]?x \rightarrow Skip$ **end**,

which is GenericContent (P, [$\mathbb{N}$ *]). Thus, P [* $\mathbb{N}$ *] equals GenericContent (P, [* $\mathbb{N}$ *]).*

Definition B.12. ***UsedC** is a function that gets a Circus action as parameter and returns a set of channels that are used within that action. For Example:*
UsedC $(\mu\ X \bullet (turnOn \rightarrow X) \sqcap (turnOff \rightarrow X)) = \{turnOn, turnOff\}$

This definition was created on [6].

Definition B.13. ***initials** is a function that gets a Circus action as parameter and returns a set of events that are initially offered by the action. For Example:*
initials $(turnOn \rightarrow Skip) = \{turnOn\}$
initials $(turnOn \rightarrow Skip \ \Box\ turnOff \rightarrow Skip) = \{turnOn, turnOff\}$

This definition was created on [6].

Definition B.14. ***Refinement Definition:***
$P \sqsubseteq Q = [Q \Rightarrow P] = \forall\ v : FREEV\ .\ Q \Rightarrow P$

On the expression above, the FREEV set represents the set of free (non-quantified) variables on the expression between brackets. The brackets mean universal quantification on the free-variables.

The definition above lies on [4, 6]

Definition B.15. ***Reactive Skip***
$II_{rea} = (\neg\ ok \wedge tr \leq tr') \vee (ok' \wedge tr' = tr \wedge wait' = wait \wedge ref' = ref \wedge v' = v)$

This definition was quoted from [6].

Definition B.16. ***State Assignment***

A Configuration State assignment is defined on [3] as a total function that maps the programming variables in the alphabet to (possibly loose) constants. That leads us to the formula

$(x := c) = (x' = c \land v' = v)$,

where c is a (possibly loose) constant (the value of c is determined as an expression that appears on the constraint of the configuration). The v' = v factor (v is the set of all non-used variables by the assignment) comes from the fact that the function is total, and all the variables on the state space that were not used by the assignment (including the observational variables ok, tr, wait and ref) must have their values mapped to their current values. A consequence of this description is that another possible definition is

$(x := c) = x' = c \land wait' = wait \land ok' = ok \land tr' = tr \land ref' = ref \land nalocalvars' = nalocalvars$,

Where nalocalvars is the set of all program (non-observational) variables (except x, because x lies on the assignment) of the state space. We can generalise the definition above for any assignment s using a ***factor f*** *such that, for example, if*

$s = (x, y := 0, 1)$

then $\mathbf{f} = \mathbf{f}(\alpha, \alpha') = (x' = 0 \land y' = 1)$.

So, for any Configuration State Assignment s,

$s = s\ (\alpha, \alpha') = f \land wait' = wait \land ok' = ok \land tr' = tr \land ref' = ref \land nalocalvars' = nalocalvars$.

Definition B.17. ***Reactive Design Assignment***

$v :=_{RD} e = Lift\ (v := e)\ ;\ Skip$

This definition was created on [3].

Definition B.18. ***Conditional***

$P \lhd b \rhd Q = (b \wedge P) \vee (\neg\, b \wedge Q)$

This definition was quoted from document [8].

Definition B.19. ***R1***

$R1\ (P) = P \wedge tr \leq tr'$

This definition lies on documents [3, 6].

Definition B.20. ***R2***

$R2\ (P\ (tr, tr')) = P\ (\langle\rangle, tr' - tr) = P\ [\langle\rangle, (tr' - tr) / tr\ , tr']$

This definition lies on document [6]..

Definition B.21. ***R3***

$R3\ (P) = II_{rea} \lhd wait \rhd P$

This definition lies on documents [3, 6]. The Conditional operator is defined on B.18 and II_{rea} (Reactive Skip) is defined on B.15.

Definition B.22. ***R***

$R\ (P) = R1 \circ R2 \circ R3\ (P)$

This definition lies on document [6]..

Definition B.23. ***Design***

$P \vdash Q = (P \wedge ok) \Rightarrow (Q \wedge ok')$

This definition lies on document [6, 8].

Definition B.24. ***Lift***

$Lift\ (s) = R1 \circ R3\ (true \vdash s \wedge tr' = tr \wedge \neg\ wait')$

This definition is original from the document [3]. All definitions, lemmas, theorems, laws and rules from CML can be used for *Circus* [5].

Definition B.25. ***Silent Transition**:*

$(c_1 \mid s_1 \models A_1) \xrightarrow{\tau} (c_2 \mid s_2 \models A_2)$

=

$\forall\, w \,.\, c_1 \wedge c_2 \Rightarrow Lift\ (s_1)\ ;\ A_1 \sqsubseteq Lift\ (s_2)\ ;\ A_2$

This definition is original from the document [3]. All definitions, lemmas, theorems, laws and rules from CML can be used for *Circus* [5].

Definition B.26. ***Labelled Transition**:*

$(c_1 \mid s_1 \models A_1) \xrightarrow{l} (c_2 \mid s_2 \models A_2)$

=

$\forall\, w \,.\, c_1 \wedge c_2 \Rightarrow Lift\ (s_1)\ ;\ A_1 \sqsubseteq (Lift\ (s_2)\ ;\ l \rightarrow A_2) \,\Box\, (Lift\ (s_1)\ ;\ A_1)$

This definition is original from the document [3]. All definitions, lemmas, theorems, laws and rules from CML can be used for Circus [5].

Definition B.27. ***Syntactic Transition (for processes)**:*

$(c_1 \models P_1) \rightsquigarrow (c_2 \models P_2)$

=

$$\forall\, w \,.\, c_1 \wedge c_2 \Rightarrow \left(\begin{array}{l} (\ Lift\,(getAssignments(P_1))\,;\ P_1 \sqsubseteq Lift\,(getAssignments\,(P_2))\,;\ P_2\ ) \\ \wedge\ Lift\,(getAssignments\,(P_1)) = Lift\,(getAssignments\,(P_2)) \end{array} \right)$$

$(c_1 \models P_1) \rightsquigarrow (c_2 \mid s \models BasicProc)$

=

$$\forall\, w \,.\, c_1 \wedge c_2 \Rightarrow \left(\begin{array}{l} ((Lift(getAssignments(P_1))\,;\ P_1 \sqsubseteq Lift(s);\ BasicProc)) \\ \quad \wedge\ Lift(getAssignments(P_1)) = Lift(s) \end{array} \right)$$

The definition of the getAssignments function is on B.1.

The definition of Syntactic Transition B.27 was created on this document.

Definitions B.25, B.26 and B.27 define advances on the program. These advances are refinements between *Circus* denotational expressions: when these refinement expressions are proved correct with respect to the theory of *Circus* on the Unifying Theories of Programming (UTP), it means that the advances of the program whose definitions are these refinement expressions are correct and satisfy the theory.

Definition B.28. ***let** [3]*
***let** $x \bullet A = A$*

This definition is original from the document [3]. All definitions, lemmas, theorems, laws and rules from CML can be used for *Circus* [5].

Definition B.29. ***loc** [3]*
***loc** $s \bullet P = Lift\ (s)\ ;\ P$*

This definition is original from the document [3]. All definitions, lemmas, theorems, laws and rules from CML can be used for *Circus* [5].

Definition B.30. ***Extra Choice***
$A_1 \boxplus A_2 = A_1 \Box A_2$

This definition is original from the document [3]. All definitions, lemmas, theorems, laws and rules from CML can be used for *Circus* [5]. The Extra Choice symbol ($\boxplus$) appears on the rules of External Choice on the Operational Semantics of [3] as ”[+]”.

Definition B.31. ***Extra Statement***
$V : [Pre, Post] = V\ [:]\ [Pre, Post]$

This definition was created on this thesis. The role of this definition is to represent the intermediate state between the initial state of the Specification Statement (the one in which the Program Text equals V : [Pre, Post]) and an intermediate state where precondition Pre is a factor on the constraint of the final state (the one in which the Program Text equals V [:] [Pre, Post]). This behaviour lies on Attached Rule 30.

Definition B.32. ***doC***
$doC\ (l, C) = (tr' = tr \land (l, C) \notin ref' \lhd wait' \rhd tr' = tr \frown \langle (l, C) \rangle)$

Where

- *l is a channel;*
- *C is a communication;*
- $\langle (l, C) \rangle$ *is a trace that is concatenated to tr;*

This function is used on the Denotational definition of the prefixing operator.

The definition was created on [6].

Definition B.33. $A\ ;\ B = \exists v0\ .\ A\ [v, v0] \land B\ [v0, v']$

The above definition was made on [4] and is referenced on [6].

Definition B.34. ***UTP Variable Declaration***
$(var\ x);\ P = \exists\ x\ .\ P$
The definition was created on [8].

Definition B.35. ***Substitution**:*
$A\,{}^{b}_{c} = A\ [b\ /\ ok'][c\ /\ wait]$

The definition lies on [6]. It is referenced on the denotational definition of other constructs of *Circus*, like If-Guarded Command A.5. When Substitution is referenced, it has factors as $A\,{}^{f}_{f}$ and $A\,{}^{t}_{f}$. On these cases, f means **false** and t means **true**.

Definition B.36. ***Disjunction Reactive Designs**:*
$R\ (P1 \vdash Q1) \lor R\ (P2 \vdash Q2) = R\ (P1 \land P2 \vdash P1 \lor P2)$

The definition lies on [7].

Definition B.37. ***External Choice Denotational Definition**:*
$A_1 \Box A_2 = R\ ((\neg\ A_1\,{}^{f}_{f} \land \neg\ A_2\,{}^{f}_{f}) \vdash ((A_1\,{}^{t}_{f} \land A_2\,{}^{t}_{f}) \lhd tr' = tr \land wait' \rhd (A_1\,{}^{t}_{f} \lor A_2\,{}^{t}_{f})))$

The definition lies on [6].

Definition B.38. ***CSP1**:*
$CSP1\ (P) = P \lor (\neg\ ok \land tr \leq tr')$

The definition lies on [6].

Definition B.39. ***CSP2**:*
CSP2 (P) = P ; ((ok ⇒ ok') ∧ tr' = tr ∧ wait' = wait ∧ ref' = ref ∧ v' = v)

The definition lies on [6].

Definition B.40. ***CSP3**:*
CSP3 (P) = Skip ; P

The definition lies on [6].

B.2.2 Auxiliary Lemmas

Lemma B.1. ***Refinement Healthiness Conditions***
(II_{rea} ◁ wait ▷ X) (⟨⟩, tr' - tr)
⊑ (II_{rea} ◁ wait ▷ false) (⟨⟩, tr' - tr)
Proof
For wait = true
((II_{rea}) (⟨⟩, tr' - tr)
⊑ (II_{rea}) (⟨⟩, tr' - tr))
= [Refinement Equal Sides 1]
true
For wait = false
X (⟨⟩, tr' - tr)
⊑ (false) (⟨⟩, tr' - tr)
= [Predicate Calculus]
X (⟨⟩, tr' - tr) ⊑ false
= [Refinement Definition B.14 and Predicate Calculus]
true □

Lemma B.2. ***Refinement Healthiness Conditions with R1***

$R(X) \sqsubseteq R(false)$

Proof:

$R(X) \sqsubseteq R(false)$
$=$ ***[Definitions of R1 B.19, R2 B.20 and R3 B.21]***
$(tr \leq tr' \wedge (II_{rea} \lhd wait \rhd X)(\langle\rangle, tr' - tr))$

$\sqsubseteq (tr \leq tr' \wedge (II_{rea} \lhd wait \rhd false)(\langle\rangle, tr' - tr))$
$=$ ***[Refinement Healthiness Conditions B.1 and Refinement Monotonic 6]***
true

Lemma B.3. ***Expanded Reactive Design Refined by Reactive Ok Negation*** $(R((\neg Ap \wedge ok) \Rightarrow (Ap \wedge ok')) \sqsubseteq R(\neg ok))$

Proof:

For $ok = true$:

$(R((\neg Ap \wedge true) \Rightarrow (Ap \wedge ok')) \sqsubseteq R(\neg true))$
[Predicate Calculus]
$(R((\neg Ap \wedge true) \Rightarrow (Ap \wedge ok')) \sqsubseteq R(false))$
[Refinement Healthiness Condition with R1 B.2]
true

For $ok = false$:

$(R((\neg Ap \wedge false) \Rightarrow (Ap \wedge ok')) \sqsubseteq R(\neg false))$
$=$ ***[Predicate Calculus]***
$(R(false \Rightarrow (Ap \wedge ok')) \sqsubseteq R(true))$
$=$ ***[Predicate Calculus]***
$(R(true) \sqsubseteq R(true))$
$=$ ***[Refinement Equal Sides 1]***
true

Lemma B.4. ***Refinement Implication For Substitution Equivalence*** $(A \sqsubseteq A \,\square\, B) \Rightarrow (A \sqsubseteq B)$,

provided that $A^f_f = A^t_f$ *and* $B^f_f = B^t_f$

Proof:

Be $Ap = Ap^f_f = Ap^t_f$, *and* $Bp = Bp^f_f = Bp^t_f$

$(A \sqsubseteq A \,\Box\, B) \Rightarrow (A \sqsubseteq B)$
$=$ ***[External Choice Denotational Definition B.37 + Assumption]***
$(R(\neg\, Ap \vdash Ap) \sqsubseteq R((\neg\, Ap \wedge \neg\, Bp) \vdash ((Ap \wedge Bp) \lhd tr' = tr \wedge wait' \rhd (Ap \vee Bp))))$
$\Rightarrow$
$(R(\neg\, Ap \vdash Ap) \sqsubseteq R(\neg\, Bp \vdash Bp))$

For $Bp = true$:

$(R(\neg\, Ap \vdash Ap) \sqsubseteq R((\neg\, Ap \wedge \neg\, true) \vdash ((Ap \wedge true) \lhd tr' = tr \wedge wait' \rhd (Ap \vee true))))$
$\Rightarrow$
$(R(\neg\, Ap \vdash Ap) \sqsubseteq R(\neg\, true \vdash true))$
$=$ ***[Predicate Calculus]***
$(R(\neg\, Ap \vdash Ap) \sqsubseteq R(false \vdash (Ap \lhd tr' = tr \wedge wait' \rhd true)))$
$\Rightarrow$
$(R(\neg\, Ap \vdash Ap) \sqsubseteq R(false \vdash true))$
$=$ ***[Predicate Calculus]***
$(R(\neg\, Ap \vdash Ap) \sqsubseteq R(false \vdash (Ap \lhd tr' = tr \wedge wait' \rhd true)))$
$\Rightarrow$
$(R(\neg\, Ap \vdash Ap) \sqsubseteq R(false))$
$=$ ***[Refinement Healthiness Condition with R1 B.2]***
$(R(\neg\, Ap \vdash Ap) \sqsubseteq R(false \vdash (Ap \lhd tr' = tr \wedge wait' \rhd true)))$
$\Rightarrow$
$true$
$=$ ***[Predicate Calculus]***
true

For $Bp = false$:

$(R(\neg\, Ap \vdash Ap) \sqsubseteq R((\neg\, Ap \wedge \neg\, false) \vdash ((Ap \wedge false) \lhd tr' = tr \wedge wait' \rhd (Ap \vee false))))$
$\Rightarrow$
$(R(\neg\, Ap \vdash Ap) \sqsubseteq R(\neg\, false \vdash false))$
$=$ ***[Predicate Calculus]***
$(R(\neg\, Ap \vdash Ap) \sqsubseteq R((\neg\, Ap \wedge \neg\, false) \vdash ((Ap \wedge false) \lhd tr' = tr \wedge wait' \rhd (Ap \vee false))))$
$\Rightarrow$
$(R(\neg\, Ap \vdash Ap) \sqsubseteq R(\neg\, false \vdash false))$

$= $ ***[Predicate Calculus]***
$(R(\neg\, Ap \vdash Ap) \sqsubseteq R((\neg\, Ap \wedge true) \vdash ((Ap \wedge false) \lhd tr' = tr \wedge wait' \rhd (Ap \vee false))))$
$\Rightarrow$
$(R(\neg\, Ap \vdash Ap) \sqsubseteq R(true \vdash false)$
$= $ ***[Predicate Calculus]***
$(R(\neg\, Ap \vdash Ap) \sqsubseteq R(\neg\, Ap \vdash (false \lhd tr' = tr \wedge wait' \rhd Ap)))$
$\Rightarrow$
$(R(\neg\, Ap \vdash Ap) \sqsubseteq R(true \vdash false))$
$= $ ***[Definition of Conditional B.18 + Predicate Calculus]***
$(R(\neg\, Ap \vdash Ap) \sqsubseteq R(\neg\, Ap \vdash (\neg\, (tr' = tr \wedge wait') \wedge Ap)))$
$\Rightarrow$
$(R(\neg\, Ap \vdash Ap) \sqsubseteq R(true \vdash false))$
$= $ ***[Design Definition B.23]***
$(R(\neg\, Ap \vdash Ap) \sqsubseteq R(\neg\, Ap \vdash (\neg\, (tr' = tr \wedge wait') \wedge Ap)))$
$\Rightarrow$
$(R((\neg\, Ap \wedge ok) \Rightarrow (Ap \wedge ok')) \sqsubseteq R(\neg\, ok))$
$= $ ***[Expanded Reactive Design Refined by Reactive Ok Negation B.3]***
$(R(\neg\, Ap \vdash Ap) \sqsubseteq R(\neg\, Ap \vdash (\neg\, (tr' = tr \wedge wait') \wedge Ap)))$
$\Rightarrow$
$true$
$= $ ***[Predicate Calculus]***
$true$

Lemma B.5. ***Lift Substitution Equivalence***
$(Lift(s))^{\,f}_{\,f} = (Lift(s))^{\,t}_{\,f}$

Proof:

$(Lift(s))^{\,f}_{\,f}$
$= $ ***[Definition of Lift B.24]***
$(R1 \circ R3(true \vdash s \wedge tr' = tr \wedge \neg\, wait'))^{\,f}_{\,f}$
$= $ ***[Definitions of R1 B.19 and R3 B.21]***
$(tr \leq tr' \wedge (II_{rea} \lhd wait \rhd (true \vdash s \wedge tr' = tr \wedge \neg\, wait')))^{\,f}_{\,f}$
$= $ ***[Substitution B.35]***
$tr \leq tr' \wedge (II_{rea}{}^{\,f}_{\,f} \lhd false \rhd (true \vdash s \wedge tr' = tr \wedge \neg\, wait'))$
$= $ ***[Definition of Conditional B.18 and Predicate Calculus]***
$tr \leq tr' \wedge (true \vdash s \wedge tr' = tr \wedge \neg\, wait')$

$= $ ***[Definition of Conditional B.18 and Predicate Calculus]***
$tr \leq tr' \wedge (II_{rea}\,{}^{t}_{f} \lhd false \rhd (true \vdash s \wedge tr' = tr \wedge \neg\, wait'))$
$= $ ***[Substitution B.35]***
$tr \leq tr' \wedge (II_{rea} \lhd wait \rhd (true \vdash s \wedge tr' = tr \wedge \neg\, wait'))\,{}^{t}_{f}$
$= $ ***[Definitions of R1 B.19 and R3 B.21]***
$(R1 \circ R3(true \vdash s \wedge tr' = tr \wedge \neg\, wait'))\,{}^{t}_{f}$
$= $ ***[Definition of Lift B.24]***
$(Lift(s))\,{}^{t}_{f}$

Lemma B.6. ***Healthy True Refinement:*** $\forall\, x\,.\; R\,(true) \sqsubseteq R(x)$

Proof
$\forall\, x\,.\; R\,(true) \sqsubseteq R(x)$
= [Quantifier $\forall$ ***x can be omitted]***
$R\,(true) \sqsubseteq R(x)$

For x = true:
$(R\,(true) \sqsubseteq R(true))$
= [Refinement Equal Sides 1]
true

For x = false:
$(R\,(true) \sqsubseteq R(false))$
= [Definitions of R1 B.19, R2 B.20 and R3 B.21]
$tr \leq tr' \wedge (II_{rea} \lhd wait \rhd true)\ (\langle\rangle,\ tr' - tr)$
$\sqsubseteq tr \leq tr' \wedge (II_{rea} \lhd wait \rhd false)\ (\langle\rangle,\ tr' - tr)$
= [Ref. Conjunctive Monotonic B.13 and Ref. Healthiness Conditions B.1]
true

Lemma B.7. ***Design of Assignment R2:***
(true $\vdash x' = w_0 \wedge v' = v \wedge tr' = tr \wedge \neg\, wait'$*) is R2*

Proof
true $\vdash$ x' = w_0 $\wedge$ v' = v $\wedge$ tr' = tr $\wedge$ $\neg$ wait'
= [Expression does not contain tr' neither tr]
true $\vdash$ x' = w_0 $\wedge$ v' = v $\wedge$ tr' = tr $\wedge$ $\neg$ wait' [$\langle\rangle$, tr' - tr / tr , tr']
= [Predicate Calculus]

true ⊢ x' = w_0 ∧ v' = v ∧ tr' = tr ∧ ¬ wait' (⟨⟩, tr' - tr)
= **[Definition of R2 B.20]**
R2 (true ⊢ x' = w_0 ∧ v' = v ∧ tr' = tr ∧ ¬ wait')
□

Lemma B.8. *(Lift (var x ; x := w_0) ; A) ⇒ (**var** x : T • A),*

provided that

- w_0 : T

Proof
(Lift (var x ; x := w_0) ; A)
= [Lift CSP4 B.2.26]
(Lift (var x ; x := w_0) ; Skip ; A)
= [Definition of Lift B.24]
(R1 ∘ R3 (true ⊢ var x ; x := w_0 ∧ tr' = tr ∧ ¬ wait') ; Skip ; A)
= [Design of Assignment R2 B.7]
(R1 ∘ R3 ∘ R2 (true ⊢ var x ; x := w_0 ∧ tr' = tr ∧ ¬ wait') ; Skip ; A)
= [R2 and R3 Composition Commutative B.2.49]
(R1 ∘ R2 ∘ R3 (true ⊢ var x ; x := w_0 ∧ tr' = tr ∧ ¬ wait') ; Skip ; A)
= [Definition of R B.22]
(R (true ⊢ var x ; x := w_0 ∧ tr' = tr ∧ ¬ wait') ; Skip ; A)
= [State Assignment B.16]
(R (true ⊢ var x ; x' = w_0 ∧ v' = v ∧ tr' = tr ∧ ¬ wait') ; Skip ; A)
= [UTP Variable Declaration B.34]
(R (true ⊢ ∃ x . x' = w_0 ∧ v' = v ∧ tr' = tr ∧ ¬ wait') ; Skip ; A)
= [Assumption w_0 : T]
(R (true ⊢ ∃ x . x' = w_0 ∧ w_0 : T ∧ v' = v ∧ tr' = tr ∧ ¬ wait') ; Skip ; A)
= [Predicate Calculus: x' = w_0 ∧ w_0 : T ⇒ x : T]
(R (true ⊢ ∃ x . x' = w_0 ∧ w_0 : T ∧ x : T ∧ v' = v ∧ tr' = tr ∧ ¬ wait') ; Skip ; A)
⇒ [Predicate Calculus: x' = w_0 ∧ w_0 : T ∧ x : T ⇒ ∃ x, x' : T]
(R (true ⊢ ∃ x, x' : T . x' = w_0 ∧ w_0 : T ∧ x : T ∧ v' = v ∧ tr' = tr ∧ ¬ wait') ; Skip ; A)
= [Predicate Calculus: x, x' and w_0 do not occur on R]
∃ x, x' . x' = w_0 ∧ w_0 : T ∧ x : T ∧ (R (true ⊢ v' = v ∧ tr' = tr ∧ ¬ wait') ; Skip ; A)
⇒ [Predicate Calculus: removing x' = w_0, w_0 : T and x : T weakens the expression]
∃ x, x' : T. (R (true ⊢ v' = v ∧ tr' = tr ∧ ¬ wait') ; Skip ; A)
= [Skip denotational definition A.3]

$\exists\, x, x' : T.\ (Skip\ ;\ Skip\ ;\ A)$
= [Theorem B.2.2 twice]
$\exists\, x, x' : T\ .\ A$
= [Variable Block Denotational Definition A.1]
$(\textbf{var}\ x : T \bullet A)$ □

Lemma B.9.
$(c_1 \models P_1\ OP\ P_2) \rightsquigarrow (c_2 \models P_3\ OP\ P_2) =$
$c_1 \wedge c_2 \Rightarrow$
$(Lift\,(gA\,(P_1)) \wedge Lift\,(gA\,(P_2));\ P_1\ OP\ P_2$
$\sqsubseteq (Lift\,(gA\,(P_1)) \wedge Lift\,(gA\,(P_2));\ P_3\ OP\ P_2))$

provided that

$Lift\ (gA\ (P_1)) = Lift\ (gA\ (P_3))$
Proof:
To reduce verbose, we create the abbreviation $gA(s)$ *such that*
$gA\,(s) = getAssignments\,(s)$

LHS
=
$(c_1 \models P_1\ OP\ P_2) \rightsquigarrow (c_2 \models P_3\ OP\ P_2)$
= [Definition of Syntactic Transition B.27]
$\forall\, w\,.\, c_1 \wedge c_2 \Rightarrow$
$$\left(\begin{array}{l}((Lift\,(gA\,(P_1\ OP\ P_2));\ P_1\ OP\ P_2 \sqsubseteq Lift\,(gA\,(P_3\ OP\ P_2));\ P_3\ OP\ P_2))\\ \wedge\ Lift\,(gA\,(P_1\ OP\ P_2)) = Lift\,(gA\,(P_3\ OP\ P_2))\end{array}\right)$$
= [w is universally quantified, so we abstract it]
$c_1 \wedge c_2 \Rightarrow$
$$\left(\begin{array}{l}((Lift\,(gA\,(P_1\ OP\ P_2));\ P_1\ OP\ P_2 \sqsubseteq Lift\,(gA\,(P_3\ OP\ P_2));\ P_3\ OP\ P_2))\\ \wedge\ Lift\,(gA\,(P_1\ OP\ P_2)) = Lift\,(gA\,(P_3\ OP\ P_2))\end{array}\right)$$
= [Definition of getAssignments B.1 (4x) for binary Operator OP]
$c_1 \wedge c_2 \Rightarrow$
$$\left(\begin{array}{l}((Lift\,(gA\,(P_1) \wedge gA\,(P_2));\ P_1\ OP\ P_2 \sqsubseteq Lift\,(gA\,(P_3) \wedge gA\,(P_2));\ P_3\ OP\ P_2))\\ \wedge\ Lift\,(gA\,(P_1) \wedge gA\,(P_2)) = Lift\,(gA\,(P_3) \wedge gA\,(P_2))\end{array}\right)$$

= [Lift And B.2.48]

$$c_1 \wedge c_2 \Rightarrow \left(\begin{array}{l} ((Lift\,(gA\,(P_1) \wedge gA\,(P_2))\,;\ P_1\ OP\ P_2 \sqsubseteq\ Lift\,(gA\,(P_3) \wedge gA\,(P_2))\,;\ P_3\ OP\ P_2)) \\ \wedge\ Lift\,(gA\,(P_1)) \wedge Lift\,(gA\,(P_2)) = Lift\,(gA\,(P_3)) \wedge Lift\,(gA\,(P_2)) \end{array} \right)$$

= [Assumption]

$$c_1 \wedge c_2 \Rightarrow \left(\begin{array}{l} ((Lift\,(gA\,(P_1) \wedge gA\,(P_2))\,;\ P_1\ OP\ P_2 \sqsubseteq Lift\,(gA\,(P_3) \wedge gA\,(P_2))\,;\ P_3\ OP\ P_2)) \\ \wedge\ Lift\,(gA\,(P_3)) \wedge Lift\,(gA\,(P_2)) = Lift\,(gA\,(P_3)) \wedge Lift\,(gA\,(P_2)) \end{array} \right)$$

= [Predicate Calculus: (P = P) = true]

$$c_1 \wedge c_2 \Rightarrow \left(\begin{array}{l} ((Lift\,(gA\,(P_1) \wedge gA\,(P_2))\,;\ P_1\ OP\ P_2 \sqsubseteq Lift\,(gA\,(P_3) \wedge gA\,(P_2))\,;\ P_3\ OP\ P_2)) \\ \wedge\ true \end{array} \right)$$

= [Predicate Calculus]

$$c_1 \wedge c_2 \Rightarrow \left(((Lift\,(gA\,(P_1) \wedge gA\,(P_2))\,;\ P_1\ OP\ P_2 \sqsubseteq Lift\,(gA\,(P_3) \wedge gA\,(P_2))\,;\ P_3\ OP\ P_2)) \right)$$

= [Lift And B.2.48]

$$c_1 \wedge c_2 \Rightarrow (Lift\,(gA\,(P_1)) \wedge Lift\,(gA\,(P_2))\,;\ P_1\ OP\ P_2 \sqsubseteq (Lift\,(gA\,(P_3)) \wedge Lift\,(gA\,(P_2))\,;\ P_3\ OP\ P_2))$$

= [Assumption]

$$c_1 \wedge c_2 \Rightarrow (Lift\,(gA\,(P_1)) \wedge Lift\,(gA\,(P_2))\,;\ P_1\ OP\ P_2 \sqsubseteq (Lift\,(gA\,(P_1)) \wedge Lift\,(gA\,(P_2))\,;\ P_3\ OP\ P_2))$$

=

RHS

Lemma B.10. $(c_1 \models P_1\ OP\ P_2) \rightsquigarrow (c_2 \models P_1\,OP\,P_3)$ *=*

$$c_1 \wedge c_2 \Rightarrow \left((Lift\,(gA\,(P_1)) \wedge Lift\,(gA\,(P_2))\,;\ P_1\ OP\ P_2 \sqsubseteq (Lift\,(gA\,(P_1)) \wedge Lift\,(gA\,(P_2))\,;\ P_1\,OP\,P_3)) \right)$$

provided that Lift (gA (P_2)*) = Lift (gA* (P_3)*)*

Proof:

To reduce verbose, we create the abbreviation $gA(s)$ *such that*

$gA(s) = getAssignments(s)$

LHS

=

$(c_1 \models P_1\ OP\ P_2) \rightsquigarrow (c_2 \models P_1\, OP P_3)$

= [Definition of Syntactic Transition B.27]

$$\forall w \,.\, c_1 \wedge c_2 \Rightarrow \left(\begin{array}{l} ((Lift\,(gA\,(P_1\ OP\ P_2))\,;\ P_1\ OP\ P_2 \sqsubseteq Lift\,(gA\,(P_1\, OP P_3))\,;\ P_1\, OP P_3)) \\ \wedge\ Lift\,(gA\,(P_1\ OP\ P_2)) = Lift\,(gA\,(P_1\, OP P_3)) \end{array} \right)$$

= [w is universally quantified, so we abstract it]

$$c_1 \wedge c_2 \Rightarrow \left(\begin{array}{l} ((Lift\,(gA\,(P_1\ OP\ P_2))\,;\ P_1\ OP\ P_2 \sqsubseteq Lift\,(gA\,(P_1\, OP P_3))\,;\ P_1\, OP P_3)) \\ \wedge\ Lift\,(gA\,(P_1\ OP\ P_2)) = Lift\,(gA\,(P_1\, OP P_3)) \end{array} \right)$$

= [Definition of getAssignments B.1]

$$c_1 \wedge c_2 \Rightarrow \left(\begin{array}{l} ((Lift\,(gA\,(P_1) \wedge gA\,(P_2))\,;\ P_1\ OP\ P_2 \sqsubseteq Lift\,(gA\,(P_1) \wedge gA\,(P_3))\,;\ P_1\, OP P_3)) \\ \wedge\ Lift\,(gA\,(P_1) \wedge gA\,(P_2)) = Lift\,(gA\,(P_1) \wedge gA\,(P_3)) \end{array} \right)$$

= [Lift And B.2.48]

$$c_1 \wedge c_2 \Rightarrow \left(\begin{array}{l} ((Lift\,(gA\,(P_1) \wedge gA\,(P_2))\,;\ P_1\ OP\ P_2 \sqsubseteq Lift\,(gA\,(P_1) \wedge gA\,(P_3))\,;\ P_1\, OP P_3)) \\ \wedge\ Lift\,(gA\,(P_1)) \wedge Lift\,(gA\,(P_2)) = Lift\,(gA\,(P_1)) \wedge Lift\,(gA\,(P_3)) \end{array} \right)$$

= [Assumption]

$$c_1 \wedge c_2 \Rightarrow \left(\begin{array}{l} ((Lift\,(gA\,(P_1) \wedge gA\,(P_2))\,;\ P_1\ OP\ P_2 \sqsubseteq Lift\,(gA\,(P_1) \wedge gA\,(P_3))\,;\ P_1\, OP P_3)) \\ \wedge\ Lift\,(gA\,(P_1)) \wedge Lift\,(gA\,(P_3)) = Lift\,(gA\,(P_1)) \wedge Lift\,(gA\,(P_3)) \end{array} \right)$$

= [Predicate Calculus: (P = P) = true]

$$c_1 \wedge c_2 \Rightarrow \left(\begin{array}{l} ((Lift\,(gA\,(P_1) \wedge gA\,(P_2))\,;\ P_1\ OP\ P_2 \sqsubseteq Lift\,(gA\,(P_1) \wedge gA\,(P_3))\,;\ P_1\, OP P_3)) \\ \wedge\ true \end{array} \right)$$

= [Predicate Calculus: (P ∧ true) = P]

$$c_1 \wedge c_2 \Rightarrow \left(\begin{array}{l} ((Lift\,(gA\,(P_1) \wedge gA\,(P_2))\,;\ P_1\ OP\ P_2 \sqsubseteq Lift\,(gA\,(P_1) \wedge gA\,(P_3))\,;\ P_1\, OP P_3)) \end{array} \right)$$

= [Lift And B.2.48]

$$c_1 \wedge c_2 \Rightarrow \left(\begin{array}{l} (Lift\,(gA\,(P_1)) \wedge Lift\,(gA\,(P_2))\,;\ P_1\ OP\ P_2 \sqsubseteq (Lift\,(gA\,(P_1)) \wedge Lift\,(gA\,(P_3))\,;\ P_1\, OP P_3)) \end{array} \right)$$

= [Assumption]

$$c_1 \wedge c_2 \Rightarrow$$

$$\left(\quad (Lift\,(gA\,(P_1)) \wedge Lift\,(gA\,(P_2))\,;\ P_1\ OP\ P_2 \sqsubseteq (Lift\,(gA\,(P_1)) \wedge Lift\,(gA\,(P_2))\,;\ P_1\,OP\,P_3)) \right)$$
$=$
RHS

Lemma B.11. $true \sqsubseteq X$

Proof

$true \sqsubseteq X$
$= [X \Rightarrow true]$
$= [true] = true$
□

Lemma B.12. **if** $(pred_1) \rightarrow A_1 \,[\!]\, ... \,[\!]\, (pred_n) \rightarrow A_n$ **fi** $\sqsubseteq A1$

provided that

- $pred_1$

***Proof**:*

if $(pred_1) \rightarrow A_1 \,[\!]\, ... \,[\!]\, (pred_n) \rightarrow A_n$ **fi** $\sqsubseteq A1$
= [If-Guarded Command Denotational Definition A.5]
R (
$(pred_1 \vee ... \vee pred_n)$
$\wedge (pred_1 \Rightarrow \neg\ {A_1}^{\,f}_{\,f} \wedge ... \wedge pred_n \Rightarrow \neg\ {A_n}^{\,f}_{\,f}) \vdash (({pred_1} \wedge {A_1}^{\,t}_{\,f}) \vee ... \vee (pred_n \wedge {A_n}^{\,t}_{\,f})$
$)) \sqsubseteq A1$
= [Assumption $pred_1$]
$(R$ (
$(true \vee ... \vee pred_n)$
$\wedge (true \Rightarrow \neg\ {A_1}^{\,f}_{\,f} \wedge ... \wedge pred_n \Rightarrow \neg\ {A_n}^{\,f}_{\,f}) \vdash (({true} \wedge {A_1}^{\,t}_{\,f}) \vee ... \vee (pred_n \wedge {A_n}^{\,t}_{\,f})$
$))) \sqsubseteq A1$
= [Predicate Calculus]
$(R$ (
$true$
$\wedge (true \Rightarrow \neg\ {A_1}^{\,f}_{\,f} \wedge ... \wedge pred_n \Rightarrow \neg\ {A_n}^{\,f}_{\,f}) \vdash (({true} \wedge {A_1}^{\,t}_{\,f}) \vee ... \vee (pred_n \wedge {A_n}^{\,t}_{\,f})$
$))) \sqsubseteq A1$
= [Predicate Calculus]

$(R\ ((true \Rightarrow \neg\ {A_1}_f^f \wedge ... \wedge pred_n \Rightarrow \neg\ {A_n}_f^f) \vdash ((true \wedge {A_1}_f^t) \vee ... \vee (pred_n \wedge {A_n}_f^t)$
$))) \sqsubseteq A1$

= [Predicate Calculus]

$(R\ ((\neg\ {A_1}_f^f \wedge ... \wedge pred_n \Rightarrow \neg\ {A_n}_f^f) \vdash ({A_1}_f^t \vee ... \vee (pred_n \wedge {A_n}_f^t)$
$))) \sqsubseteq A1$

= [Disjunction Reactive Designs B.36]

$(R\ (\neg\ {A_1}_f^f \vdash {A_1}_f^t) \vee R\ (\neg\ (pred_2 \Rightarrow {A_2}_f^f) \vdash (pred_2 \wedge {A_2}_f^t)) ... \vee R\ (\neg\ (pred_2 \Rightarrow {A_2}_f^f) \vdash$
$(pred_2 \wedge {A_2}_f^t))$
$) \sqsubseteq A1$

= [Refinement Definition B.14]

$$\left[\begin{array}{l} A1 \Rightarrow \\ \left(\begin{array}{l} R(\neg\, {A_1}_f^f \vdash {A_1}_f^t) \vee R(\neg\, (pred_2 \Rightarrow {A_2}_f^f) \vdash (pred_2 \wedge {A_2}_f^t)) \vee ... \\ \vee R(\neg\, (pred_n \Rightarrow {A_n}_f^f) \vdash (pred_n \wedge {A_n}_f^t)) \end{array}\right) \end{array}\right]$$

= [CSP process as a Self-Reactive Design 12, having, from theorem 13, that it can be applied to Circus Actions]

$$\left[\begin{array}{l} A1 \Rightarrow \\ \left(\begin{array}{l} A1 \vee R(\neg\, (pred_2 \Rightarrow {A_2}_f^f) \vdash (pred_2 \wedge {A_2}_f^t)) \vee ... \\ \vee R(\neg\, (pred_n \Rightarrow {A_n}_f^f) \vdash (pred_n \wedge {A_n}_f^t)) \end{array}\right) \end{array}\right]$$

= [Predicate Calculus]

$$\left[\begin{array}{l} \neg A1 \vee \\ \left(\begin{array}{l} A1 \vee R(\neg\, (pred_2 \Rightarrow {A_2}_f^f) \vdash (pred_2 \wedge {A_2}_f^t)) \vee ... \\ \vee R(\neg\, (pred_n \Rightarrow {A_n}_f^f) \vdash (pred_n \wedge {A_n}_f^t)) \end{array}\right) \end{array}\right]$$

= [Predicate Calculus]

$$\left[\begin{array}{l} \neg A1 \vee A1 \vee \\ \left(\begin{array}{l} R(\neg\, (pred_2 \Rightarrow {A_2}_f^f) \vdash (pred_2 \wedge {A_2}_f^t)) \vee ... \\ \vee R(\neg\, (pred_n \Rightarrow {A_n}_f^f) \vdash (pred_n \wedge {A_n}_f^t)) \end{array}\right) \end{array}\right]$$

= [Predicate Calculus]

$$\left[\begin{array}{l} true \vee \\ \left(\begin{array}{l} R(\neg\, (pred_2 \Rightarrow {A_2}_f^f) \vdash (pred_2 \wedge {A_2}_f^t)) \vee ... \\ \vee R(\neg\, (pred_n \Rightarrow {A_n}_f^f) \vdash (pred_n \wedge {A_n}_f^t)) \end{array}\right) \end{array}\right]$$

= [Predicate Calculus]

$[true]$

= [Predicate Calculus]

$true$

Lemma B.13. ***Refinement Conjunctive Monotonic***

$(P \wedge F \sqsubseteq Q \wedge F)$

provided that $(P \sqsubseteq Q)$

***Proof**:*
For F = true:
$(P \wedge true \sqsubseteq Q \wedge true)$
= [Predicate Calculus]
$(P \sqsubseteq Q)$
= [Assumption]
true

For F = false:
$(P \wedge false \sqsubseteq Q \wedge false)$
= [Predicate Calculus]
$(false \sqsubseteq false)$
= [Refinement Definition B.14]
$[(false \Rightarrow false)]$
= [Predicate Calculus]
true

Lemma B.14. ***Design Trace R2*** $(true \vdash s \wedge tr' = tr0 \wedge \neg\, wait')$ *is R2*
***Proof**:*
$(true \vdash s \wedge tr' = tr0 \wedge \neg\, wait')$
= [Sequence Property]
$(true \vdash s \wedge tr' - tr0 = \langle\rangle \wedge \neg\, wait')$
= [Definition of R2 B.20]
$R2\ (true \vdash s \wedge tr' = tr0 \wedge \neg\, wait')$

Lemma B.15. ****Existential Quantifier Shifted Inside R:***
$\exists\, x\, .\ R\,(P) = R\,(\exists\, x\, .\ P)$,

provided that x is neither tr nor wait

This theorem lies on [7].

Lemma B.16. ***Existential Quantifier Distributed Throughout Design:***
$\exists\, x \,.\, (true \vdash Post) = (true \vdash (\exists\, x \,.\, Post))$,

provided that x is neither ok or ok'.

Proof:

$\exists\, x \,.\, (true \vdash Post)$
[Definition of Design B.23]
$\exists\, x \,.\, ((true \wedge ok) \Rightarrow (Post \wedge ok'))$
[Predicate Calculus]
$\exists\, x \,.\, (ok \Rightarrow (Post \wedge ok'))$
[Predicate Calculus]
$\exists\, x \,.\, (\neg\, ok \vee (Post \wedge ok'))$
[Predicate Calculus]
$(\exists\, x \,.\, \neg\, ok) \vee (\exists\, x \,.\, Post \wedge ok')$
[Predicate Calculus]
$(\exists\, x \,.\, \neg\, (true \wedge ok)) \vee (\exists\, x \,.\, Post \wedge ok')$
[Predicate Calculus]
$(\neg\ \forall\, x \,.\, (true \wedge ok)) \vee (\exists\, x \,.\, Post \wedge ok')$
[Predicate Calculus]
$(\forall\, x \,.\, (true \wedge ok)) \Rightarrow (\exists\, x \,.\, Post \wedge ok')$
[Predicate Calculus]
$(\forall\, x \,.\, (true \wedge ok)) \Rightarrow (\exists\, x \,.\, Post \wedge ok')$
[Predicate Calculus]
$((true \wedge \forall\, x \,.\, ok)) \Rightarrow (\exists\, x \,.\, Post \wedge ok')$
[Assumption]
$((true \wedge ok)) \Rightarrow (\exists\, x \,.\, Post \wedge ok')$
[Assumption]
$((true \wedge ok)) \Rightarrow ((\exists\, x \,.\, Post) \wedge ok')$
[Definition of Design B.23]
$true \vdash (\exists\, x \,.\, Post)$

Lemma B.17. ***Labelled Transition Implication***
$(c_1 \mid s_1 \models A_1) \xrightarrow{d^* w_1} (c_3 \mid s_3 \models A_3)$
=

$\forall\ w.\ (Lift\ (s_1);\ A_1) \sqsubseteq \forall\ w.\ (Lift\ (s_3);\ d^* w_1 \rightarrow A_3)$

Proof:

$(c_1 \mid s_1 \models A_1) \xrightarrow{d^* w_1} (c_3 \mid s_3 \models A_3)$
= *[Definition of Labelled Transition B.26]*
$\forall\ w.\ c_1 \wedge c_3 \Rightarrow (Lift\ (s_1);\ A_1) \sqsubseteq (Lift\ (s_3);\ d^* w_1 \rightarrow A_3) \Box (Lift\ (s_1);\ A_1)$
= *[External Choice Refinement Implication 15]*
$\forall\ w.\ c_1 \wedge c_3 \Rightarrow (Lift\ (s_1);\ A_1) \sqsubseteq (Lift\ (s_3);\ d^* w_1 \rightarrow A_3)$

B.2.3 Auxiliary Theorems

Theorem 1. ***Refinement Equal Sides:***
$A \sqsubseteq A$

Proof
A $\sqsubseteq$ A
= [Definition of Refinement]
[A $\Rightarrow$ A]
= [Predicate Calculus]
[$\neg$ A $\vee$ A]
= [Predicate Calculus]
[true]
= [Predicate Calculus]
true
$\Box$

Theorem 2.

Silent Transition between equivalent nodes:
$(c \mid s \models A) \xrightarrow{\tau} (c \mid s \models A)$

Proof
(c | s $\models$ A) $\xrightarrow{\tau}$ (c | s $\models$ A)
= [Definition of Silent Transition, B.25]
$\forall$ w . c $\wedge$ c $\Rightarrow$ ((Lift (s) ; A $\sqsubseteq$ Lift (s) ; A)
= [Refinement Equal Sides 1]
$\forall$ w. c $\wedge$ c $\Rightarrow$ true

= [(P $\Rightarrow$ true) = true]

$\forall$ w. true

=

true

□

Theorem 3.

***Sigma Equal**:*

$(c \models P) \rightsquigarrow (c \models P)$

Proof (c $\models$ P) $\rightsquigarrow$ (c $\models$ P)

= [Definition of Sigma Transition]

$$\forall w.c \wedge c \Rightarrow \left(\begin{array}{l} (Lift(getAssignments(P));\ P \sqsubseteq Lift(getAssignments(P));\ P) \\ \wedge\ Lift(getAssignments(P)) = Lift(getAssignments(P)) \end{array} \right)$$

= [Refinement Equal Sides (2x) 1]

$\forall$ w . c $\wedge$ c $\Rightarrow$ (true $\wedge$ true)

= [Predicate Calculus]

$\forall$ w. c $\wedge$ c $\Rightarrow$ true

= [Predicate Calculus]

true □

Theorem 4.

***Sigma Equal Basic Process**:*

$(c \models P) \rightsquigarrow (c \mid s \models B)$

provided that P = B

Proof

(c $\models$ P) $\rightsquigarrow$ (c | s $\models$ B)

= [Definition of Syntactic Transition B.27]

$$\forall w.c \wedge c \Rightarrow \left(\begin{array}{l} (Lift(getAssignments(P));\ P \sqsubseteq Lift(s);\ B) \\ \wedge\ Lift(getAssignments(P)) = Lift(s) \end{array} \right)$$

= [Assumption P = B]

$$\forall w.c \wedge c \Rightarrow \left(\begin{array}{l} (Lift(getAssignments(B));\ B \sqsubseteq Lift(s);\ B) \\ \wedge\ Lift(getAssignments(B)) = Lift(s) \end{array} \right)$$

= [Definition of getAssignments B.1]

$$\forall w.c \land c \Rightarrow \left(\begin{array}{l} (Lift(s);\ B \sqsubseteq Lift(s);\ B) \\ \land\ Lift(s) = Lift(s) \end{array} \right)$$

= [Predicate Calculus]

$$\forall w.c \land c \Rightarrow \left(\begin{array}{l} (Lift(s);\ B \sqsubseteq Lift(s);\ B) \\ \land\ true \end{array} \right)$$

= [Refinement Equal Sides 1]

$$\forall w.c \land c \Rightarrow \left(\begin{array}{l} true \\ \land\ true \end{array} \right)$$

= [Predicate Calculus]

true □

Theorem 5.

***S Sequence False**:

$s\ ;\ false = false$

This theorem is quoted on [4].

Theorem 6.

***Monotonicity of Action Refinement**:

$A_1\ ;\ A_2 \sqsubseteq B_1\ ;\ B_2$ *provided* $A_1 \sqsubseteq B_1$ *and* $A_2 \sqsubseteq B_2$

Proof The proof lies on the document [7].

□

Theorem 7.

***Lift Shift**:

$l \rightarrow Lift\ (s)\ ;\ A = Lift\ (s)\ ;\ l \rightarrow A$

provided that

- *ok = ok'*

Where FV (l) are the free variables of l, and LHS (s) returns the left variables of the sequence s of assignments. We use Theorem 11 to assume that all Lifted assignments (e.g. Lift (s)) are Circus actions.

Proof

l $\rightarrow$ Lift (s) ; A

= [Prefixing Sequence B.2.5, having, from theorem 11, that Lift (s) is a *Circus* action]

l $\rightarrow$ Skip ; Lift (s) ; A

= [Definition of Lift (B.24)]

l $\rightarrow$ Skip ; (R1 $\circ$ R3 (true $\vdash$ s $\wedge$ tr' = tr $\wedge$ $\neg$ wait')) ; A

= [Sequence Denotational Definition B.33 and Prefixing Denotational Definition A.2 (where v0 is a set that contains all observational variables in an intermediate state (ok0, wait0, ref0, tr0 and so on))]

$$\exists v0. \left(\begin{array}{l} R(true \vdash doC(l,C) \wedge vars0 = vars) \\ \wedge\ (R1 \circ R3(true \vdash s(\alpha 0, \alpha') \wedge tr' = tr0 \wedge \neg\, wait')) \end{array} \right) ; A$$

= [Definition of doC B.32]

$$\exists v0. \left(\begin{array}{l} R\left(true \vdash \left(\begin{array}{l} tr0 = tr \wedge (l,C) \notin ref0 \\ \lhd wait0 \rhd \\ tr0 = tr \frown \langle (l,C) \rangle \end{array} \right) \wedge vars0 = vars \right) \\ \wedge\ (R1 \circ R3(true \vdash s(\alpha 0, \alpha') \wedge tr' = tr0 \wedge \neg\, wait')) \end{array} \right) ; A$$

= [Design Trace R2 B.14]

$$\exists v0. \left(\begin{array}{l} R\left(true \vdash \left(\begin{array}{l} tr0 = tr \wedge (l,C) \notin ref0 \\ \lhd wait0 \rhd \\ tr0 = tr \frown \langle (l,C) \rangle \end{array} \right) \wedge vars0 = vars \right) \\ \wedge\ (R(true \vdash s(\alpha 0, \alpha') \wedge tr' = tr0 \wedge \neg\, wait')) \end{array} \right) ; A$$

= [State Assignment B.16]

$$\exists v0. \left(\begin{array}{l} R\left(true \vdash \left(\begin{array}{l} tr0 = tr \wedge (l,C) \notin ref0 \\ \lhd wait0 \rhd \\ tr0 = tr \frown \langle (l,C) \rangle \end{array} \right) \wedge vars0 = vars \right) \\ \wedge\ \left(R\left(true \vdash \left(\begin{array}{l} f(\alpha 0, \alpha') \wedge wait' = wait0 \\ \wedge\ ok' = ok0 \wedge tr' = tr0 \wedge ref' = ref0 \\ \wedge\ nalocalvars' = nalocalvars0 \\ \wedge\ tr' = tr0 \wedge \neg\, wait' \end{array} \right) \right) \right) \end{array} \right) ; A$$

= [Closure Conjunctive R B.2.42]

$$\exists v0.\ \left(\begin{array}{l} R\left(\begin{array}{l}\left(\begin{array}{l} true \vdash \left(\begin{array}{l} tr0 = tr \wedge (l,C) \notin ref0 \\ \lhd wait0 \rhd \\ tr0 = tr \frown \langle (l,C) \rangle \end{array}\right) \wedge vars0 = vars \end{array}\right) \\ \wedge \left(\begin{array}{l} true \vdash \left(\begin{array}{l} f(\alpha 0, \alpha') \wedge wait' = wait0 \\ \wedge\ ok' = ok0 \wedge tr' = tr0 \\ \wedge\ ref' = ref0 \\ \wedge\ nalocalvars' = nalocalvars0 \\ \wedge\ tr' = tr0 \wedge \neg\ wait' \end{array}\right) \end{array}\right) \end{array}\right) ; A \end{array}\right)$$

= [Designs And True B.2.46]

$$\exists v0.\ \left(\begin{array}{l} R\left(\begin{array}{l} true \vdash \left(\begin{array}{l} \left(\begin{array}{l} tr0 = tr \wedge (l,C) \notin ref0 \\ \lhd wait0 \rhd \\ tr0 = tr \frown \langle (l,C) \rangle \end{array}\right) \wedge vars0 = vars \\ \wedge f(\alpha 0, \alpha') \wedge wait' = wait0 \\ \wedge\ ok' = ok0 \wedge tr' = tr0 \\ \wedge\ ref' = ref0 \\ \wedge\ nalocalvars' = nalocalvars0 \\ \wedge\ tr' = tr0 \wedge \neg\ wait' \end{array}\right) \end{array}\right) ; A \end{array}\right)$$

= [Predicate Calculus: rearranging the order of the and operands and parenthesis]

$$\exists v0.\ \left(\begin{array}{l} R\left(\begin{array}{l} true \vdash \left(\begin{array}{l} f(\alpha 0, \alpha') \wedge \\ \left(\begin{array}{l} tr0 = tr \wedge (l,C) \notin ref0 \\ \lhd wait0 \rhd \\ tr0 = tr \frown \langle (l,C) \rangle \end{array}\right) \wedge vars0 = vars \\ \wedge\ (wait' = wait0) \wedge \neg\ wait' \\ \wedge\ (ref' = ref0) \wedge (nalocalvars' = nalocalvars0) \\ \wedge\ (tr' = tr0) \\ \wedge\ (tr' = tr0) \\ \wedge\ ok' = ok0 \end{array}\right) \end{array}\right) ; A \end{array}\right)$$

= [Existential Quantifier Shifted Inside R B.15]

$$R\left(\begin{array}{l} \exists v0.\ \left(\begin{array}{l} true \vdash \left(\begin{array}{l} f(\alpha 0, \alpha') \wedge \\ \left(\begin{array}{l} tr0 = tr \wedge (l,C) \notin ref0 \\ \lhd wait0 \rhd \\ tr0 = tr \frown \langle (l,C) \rangle \end{array}\right) \wedge vars0 = vars \\ \wedge\ (wait' = wait0) \wedge \neg\ wait0 \\ \wedge\ (ref' = ref0) \wedge (nalocalvars' = nalocalvars0) \\ \wedge\ (tr' = tr0) \wedge (tr' = tr0) \\ \wedge\ ok' = ok0 \end{array}\right) \end{array}\right) ; A \end{array}\right)$$

= [Existential Quantifier Distributed Throughout Design B.16]

$$R\left(\begin{array}{l} true \vdash \exists v0. \left(\begin{array}{l} f(\alpha 0, \alpha') \wedge \\ \left(\begin{array}{l} tr0 = tr \wedge (l, C) \notin ref0 \\ \lhd wait0 \rhd \\ tr0 = tr \frown \langle (l, C) \rangle \end{array}\right) \wedge vars0 = vars \\ \wedge\ (wait' = wait0) \wedge \neg\, wait0 \\ \wedge\ (ref' = ref0) \wedge (nalocalvars' = nalocalvars0) \\ \wedge\ (tr' = tr0) \wedge (tr' = tr0) \\ \wedge\ ok' = ok0 \end{array}\right) \end{array}\right); A$$

= [Predicate Calculus: One-point-rule]

$$R\left(\begin{array}{l} true \vdash \exists v0. \left(\begin{array}{l} f(\alpha 0, \alpha') \wedge \\ \left(\begin{array}{l} tr0 = tr \wedge (l, C) \notin ref0 \\ \lhd wait0 \rhd \\ tr0 = tr \frown \langle (l, C) \rangle \end{array}\right) \wedge vars0 = vars \\ \wedge\ (wait' = wait0) \wedge \neg\, wait0 \\ \wedge\ (ref' = ref0) \wedge (nalocalvars' = nalocalvars0) \\ \wedge\ (tr' = tr0) \wedge (tr' = tr0) \\ \wedge\ ok' = ok0 \end{array}\right) \end{array}\right); A$$

= [Predicate Calculus: replacing $\neg\, wait0$]

$$R\left(\begin{array}{l} true \vdash \exists v0. \left(\begin{array}{l} f(\alpha 0, \alpha') \wedge \\ \left(\begin{array}{l} tr0 = tr \wedge (l, C) \notin ref0 \\ \lhd wait0 \rhd \\ tr0 = tr \frown \langle (l, C) \rangle \end{array}\right) \wedge vars0 = vars \\ \wedge\ (wait' = wait0) \\ \wedge\ (ref' = ref0) \wedge (nalocalvars' = nalocalvars0) \\ \wedge\ (tr' = tr0) \wedge (tr' = tr0) \\ \wedge\ ok' = ok0 \wedge \neg\, wait0 \end{array}\right) \end{array}\right); A$$

= [Predicate Calculus: One-point-rule (inserting wait = wait0)]

$$R\left(\begin{array}{l} true \vdash \exists v0. \left(\begin{array}{l} f(\alpha 0, \alpha') \wedge wait = wait0 \wedge \\ \left(\begin{array}{l} tr0 = tr \wedge (l, C) \notin ref0 \\ \lhd wait0 \rhd \\ tr0 = tr \frown \langle (l, C) \rangle \end{array}\right) \wedge vars0 = vars \\ \wedge\ (ref' = ref0) \wedge (nalocalvars' = nalocalvars0) \\ \wedge\ (tr' = tr0) \wedge (tr' = tr0) \\ \wedge\ ok' = ok0 \wedge \neg\, wait0 \end{array}\right) \end{array}\right); A$$

= [Predicate Calculus: One-point-rule (replacing wait0 by wait' on the conditional)]

$$R\left(\begin{array}{l} true \vdash \exists v0. \left(\begin{array}{l} f(\alpha 0, \alpha') \wedge wait = wait0 \wedge \\ ((tr0 = tr \wedge (l,C) \notin ref0 \lhd wait' \rhd tr0 = tr \frown \langle (l,C) \rangle) \wedge vars0 = vars) \\ \wedge (ref' = ref0) \wedge (nalocalvars' = nalocalvars0) \\ \wedge (tr' = tr0) \wedge (tr' = tr0) \\ \wedge ok' = ok0 \wedge \neg wait0 \end{array}\right) \end{array}\right); A$$

= [Predicate Calculus: One-point-rule (inserting ref = ref0)]

$$R\left(\begin{array}{l} true \vdash \exists v0. \left(\begin{array}{l} f(\alpha 0, \alpha') \wedge wait = wait0 \wedge ref = ref0 \wedge \\ ((tr0 = tr \wedge (l,C) \notin ref0 \lhd wait' \rhd tr0 = tr \frown \langle (l,C) \rangle) \wedge vars0 = vars) \\ \wedge (nalocalvars' = nalocalvars0) \\ \wedge (tr' = tr0) \wedge (tr' = tr0) \\ \wedge ok' = ok0 \wedge \neg wait0 \end{array}\right) \end{array}\right); A$$

= [Predicate Calculus: One-point-rule (replacing $\notin ref0$ by $\notin ref'$)]

$$R\left(\begin{array}{l} true \vdash \exists v0. \left(\begin{array}{l} f(\alpha 0, \alpha') \wedge wait = wait0 \wedge ref = ref0 \wedge \\ ((tr0 = tr \wedge (l,C) \notin ref' \lhd wait' \rhd tr0 = tr \frown \langle (l,C) \rangle) \wedge vars0 = vars) \\ \wedge (nalocalvars' = nalocalvars0) \\ \wedge (tr' = tr0) \wedge (tr' = tr0) \\ \wedge ok' = ok0 \wedge \neg wait0 \end{array}\right) \end{array}\right); A$$

= [Predicate Calculus: And Idempotence (tr' = tr0) and commutative]

$$R\left(\begin{array}{l} true \vdash \exists v0. \left(\begin{array}{l} f(\alpha 0, \alpha') \wedge wait = wait0 \wedge ref = ref0 \wedge \\ ((tr0 = tr \wedge (l,C) \notin ref' \lhd wait' \rhd tr0 = tr \frown \langle (l,C) \rangle) \wedge vars0 = vars) \\ \wedge (tr' = tr0) \\ \wedge (nalocalvars' = nalocalvars0) \\ \wedge ok' = ok0 \wedge \neg wait0 \end{array}\right) \end{array}\right); A$$

= [Predicate Calculus: One-point-rule (inserting tr = tr0)]

$$R\left(\begin{array}{l} true \vdash \exists v0. \left(\begin{array}{l} f(\alpha 0, \alpha') \wedge wait = wait0 \wedge ref = ref0 \wedge tr = tr0 \wedge \\ ((tr0 = tr \wedge (l,C) \notin ref' \lhd wait' \rhd tr0 = tr \frown \langle (l,C) \rangle) \wedge vars0 = vars) \\ \wedge (tr' = tr0) \\ (nalocalvars' = nalocalvars0) \\ \wedge ok' = ok0 \wedge \neg wait0 \end{array}\right) \end{array}\right); A$$

= [Predicate Calculus: One-point-rule (removing tr' = tr0)]

$$R\left(\begin{array}{l} true \vdash \exists v0. \left(\begin{array}{l} f(\alpha 0, \alpha') \wedge wait = wait0 \wedge ref = ref0 \wedge tr = tr0 \wedge \\ ((tr0 = tr \wedge (l,C) \notin ref' \lhd wait' \rhd tr0 = tr \frown \langle (l,C) \rangle) \wedge vars0 = vars) \\ \wedge (nalocalvars' = nalocalvars0) \\ \wedge ok' = ok0 \wedge \neg wait0 \end{array}\right) \end{array}\right); A$$

= [By the def. of Prefixing Denotational Definition A.2, vars0 = vars includes all local variables (some of which are the assignments of nalocalvars), so nalocalvars $\subseteq$ vars. Thus, vars0 = vars can be rewritten as f (α, $\alpha 0$) $\wedge$ nalocalvars0 = nalocalvars, such

that f (α, $\alpha 0$) does not change its values]

$$R\left(\begin{array}{l} true \vdash \exists v0.\ \left(\begin{array}{l} f(\alpha 0,\alpha') \wedge wait = wait0 \wedge ref = ref0 \wedge tr = tr0 \wedge \\ \left(\begin{array}{l} (tr' = tr \wedge (l,C) \notin ref' \lhd wait' \rhd tr' = tr \frown \langle (l,C) \rangle) \\ \wedge f(\alpha,\alpha 0) \wedge nalocalvars0 = nalocalvars \end{array}\right) \\ \wedge (nalocalvars' = nalocalvars0) \\ \wedge ok' = ok0 \wedge \neg wait0 \end{array}\right) \end{array}\right); A$$

= [Predicate Calculus: $f(\alpha 0,\alpha')$ and $f(\alpha,\alpha 0)$ side by side]

$$R\left(\begin{array}{l} true \vdash \exists v0.\ \left(\begin{array}{l} f(\alpha 0,\alpha') \wedge f(\alpha,\alpha 0) \wedge wait = wait0 \wedge ref = ref0 \wedge tr = tr0 \wedge \\ \left(\begin{array}{l} (tr' = tr \wedge (l,C) \notin ref' \lhd wait' \rhd tr' = tr \frown \langle (l,C) \rangle) \\ \wedge nalocalvars0 = nalocalvars \end{array}\right) \\ \wedge (nalocalvars' = nalocalvars0) \\ \wedge ok' = ok0 \wedge \neg wait0 \end{array}\right) \end{array}\right); A$$

= [Predicate Calculus: switching the positions of *nalocalvars0 = nalocalvars* and *nalocalvars' = nalocalvars0*]

$$R\left(\begin{array}{l} true \vdash \exists v0.\ \left(\begin{array}{l} f(\alpha 0,\alpha') \wedge f(\alpha,\alpha 0) \wedge wait = wait0 \wedge ref = ref0 \wedge tr = tr0 \wedge \\ \left(\begin{array}{l} (tr' = tr \wedge (l,C) \notin ref' \lhd wait' \rhd tr' = tr \frown \langle (l,C) \rangle) \\ \wedge (nalocalvars' = nalocalvars0) \end{array}\right) \\ \wedge nalocalvars0 = nalocalvars \\ \wedge ok' = ok0 \wedge \neg wait0 \end{array}\right) \end{array}\right); A$$

= [Assumption: ok = ok']

$$R\left(\begin{array}{l} true \vdash \exists v0.\ \left(\begin{array}{l} f(\alpha 0,\alpha') \wedge f(\alpha,\alpha 0) \wedge wait = wait0 \wedge ref = ref0 \wedge tr = tr0 \wedge \\ \left(\begin{array}{l} (tr' = tr \wedge (l,C) \notin ref' \lhd wait' \rhd tr' = tr \frown \langle (l,C) \rangle) \\ \wedge (nalocalvars' = nalocalvars0) \end{array}\right) \\ \wedge nalocalvars0 = nalocalvars \wedge ok = ok0 \wedge \neg wait0 \end{array}\right) \end{array}\right); A$$

= [Predicate Calculus: placing *nalocalvars0 = nalocalvars* $\wedge$ *ok = ok0* $\wedge$ $\neg$ *wait0* above]

$$R\left(\begin{array}{l} true \vdash \exists v0.\ \left(\begin{array}{l} nalocalvars0 = nalocalvars \wedge ok = ok0 \wedge \neg wait0 \wedge \\ f(\alpha 0,\alpha') \wedge f(\alpha,\alpha 0) \wedge wait = wait0 \wedge ref = ref0 \wedge tr = tr0 \wedge \\ \left(\begin{array}{l} (tr' = tr \wedge (l,C) \notin ref' \lhd wait' \rhd tr' = tr \frown \langle (l,C) \rangle) \\ \wedge (nalocalvars' = nalocalvars0) \end{array}\right) \end{array}\right) \end{array}\right); A$$

= [Predicate Calculus: rearranging some factors]

$$R\left(\begin{array}{l} true \vdash \exists v0.\ \left(\begin{array}{l} f(\alpha,\alpha 0) \\ \wedge ok = ok0 \wedge wait = wait0 \wedge ref = ref0 \\ \wedge nalocalvars0 = nalocalvars \wedge tr0 = tr \wedge \neg wait0 \\ \wedge f(\alpha 0,\alpha') \wedge \\ \left(\begin{array}{l} (tr' = tr \wedge (l,C) \notin ref' \lhd wait' \rhd tr' = tr \frown \langle (l,C) \rangle) \\ \wedge (nalocalvars' = nalocalvars0) \end{array}\right) \end{array}\right) \end{array}\right); A$$

= [Existential Quantifier Distributed Throughout Design B.16]

$$R\left(\begin{array}{l}\exists v0.true \vdash \left(\begin{array}{l} f(\alpha,\alpha 0) \\ \wedge\ ok = ok0 \wedge wait = wait0 \wedge ref = ref0 \\ \wedge\ nalocalvars0 = nalocalvars \wedge tr0 = tr \wedge \neg\, wait0 \\ \wedge f(\alpha 0,\alpha') \wedge \\ \left(\begin{array}{l}(tr' = tr \wedge (l,C) \notin ref' \lhd wait' \rhd tr' = tr \frown \langle (l,C)\rangle) \\ \wedge\ (nalocalvars' = nalocalvars0)\end{array}\right)\end{array}\right)\end{array}\right); A$$

= [Existential Quantifier Shifted Inside R (reverse) B.15]

$$\exists v0.R\left(\begin{array}{l} true \vdash \left(\begin{array}{l} f(\alpha,\alpha 0) \\ \wedge\ ok = ok0 \wedge wait = wait0 \wedge ref = ref0 \\ \wedge\ nalocalvars0 = nalocalvars \wedge tr0 = tr \wedge \neg\, wait0 \\ \wedge f(\alpha 0,\alpha') \wedge \\ \left(\begin{array}{l}(tr' = tr \wedge (l,C) \notin ref' \lhd wait' \rhd tr' = tr \frown \langle (l,C)\rangle) \\ \wedge\ (nalocalvars' = nalocalvars0)\end{array}\right)\end{array}\right)\end{array}\right); A$$

= [Designs And True B.2.46]

$$\exists v0.R\left(\begin{array}{l} true \vdash \left(\begin{array}{l} f(\alpha,\alpha 0) \\ \wedge\ ok = ok0 \wedge wait = wait0 \wedge ref = ref0 \\ \wedge\ nalocalvars0 = nalocalvars \wedge tr = tr0 \wedge \neg\, wait0 \end{array}\right) \\ \wedge\ true \vdash \left(\begin{array}{l} \wedge f(\alpha 0,\alpha') \wedge \\ \left(\begin{array}{l}(tr' = tr \wedge (l,C) \notin ref' \lhd wait' \rhd tr' = tr \frown \langle (l,C)\rangle) \\ \wedge\ (nalocalvars' = nalocalvars0)\end{array}\right)\end{array}\right)\end{array}\right); A$$

= [State Assignment B.16]

$$\exists v0.R\left(\begin{array}{l} true \vdash \left(\begin{array}{l} s(\alpha,\alpha 0) \wedge tr = tr0 \wedge \neg\, wait0 \end{array}\right) \\ \wedge\ true \vdash \left(\begin{array}{l} f(\alpha 0,\alpha') \wedge \\ \left(\begin{array}{l}(tr' = tr \wedge (l,C) \notin ref' \lhd wait' \rhd tr' = tr \frown \langle (l,C)\rangle) \\ \wedge\ (nalocalvars' = nalocalvars0)\end{array}\right)\end{array}\right)\end{array}\right); A$$

= [Closure Conjunctive R B.2.42]

$$\exists v0.\left(\begin{array}{l} R\left(\begin{array}{l} true \vdash \left(\begin{array}{l} s(\alpha,\alpha 0) \wedge tr = tr0 \wedge \neg\, wait0 \end{array}\right)\end{array}\right) \\ \wedge\ R\left(\begin{array}{l} true \vdash \left(\begin{array}{l} f(\alpha 0,\alpha') \wedge \\ \left(\begin{array}{l}(tr' = tr \wedge (l,C) \notin ref' \lhd wait' \rhd tr' = tr \frown \langle (l,C)\rangle) \\ \wedge\ (nalocalvars' = nalocalvars0)\end{array}\right)\end{array}\right)\end{array}\right)\end{array}\right); A$$

= [Predicate Calculus: repositioning nalocalvars' = nalocalvars0]

$$\exists v0.\left(\begin{array}{l} R\left(\begin{array}{l} true \vdash \left(\begin{array}{l} s(\alpha,\alpha 0) \wedge tr = tr0 \wedge \neg\, wait0 \end{array}\right)\end{array}\right) \\ \wedge\ R\left(\begin{array}{l} true \vdash \left(\begin{array}{l} (tr' = tr \wedge (l,C) \notin ref' \lhd wait' \rhd tr' = tr \frown \langle (l,C)\rangle) \\ \wedge f(\alpha 0,\alpha') \wedge (nalocalvars' = nalocalvars0)\end{array}\right)\end{array}\right)\end{array}\right); A$$

= [On the denotational definition of Prefixing A.2, f (α0, α′) ∧ nalocalvars' = nalocalvars0 is vars' = vars0]

$$\exists v0. \left(\begin{array}{l} R\left(\; true \vdash \left(\; s(\alpha,\alpha 0) \wedge tr = tr0 \wedge \neg \; wait0 \; \right) \right) \\ \wedge R\left(\; true \vdash \left(\begin{array}{l} (tr' = tr \wedge (l,C) \notin ref' \lhd wait' \rhd tr' = tr \frown \langle (l,C) \rangle) \\ \wedge \; vars' = vars0 \end{array} \right) \right) \end{array} \right) ; A$$

= [Definition of Lift B.24]

$$\exists v0. \left(\begin{array}{l} Lift(s)(\alpha,\alpha 0) \\ \wedge R\left(\; true \vdash \left(\begin{array}{l} (tr' = tr \wedge (l,C) \notin ref' \lhd wait' \rhd tr' = tr \frown \langle (l,C) \rangle) \\ \wedge \; vars' = vars0 \end{array} \right) \right) \end{array} \right) ; A$$

= [Definition of doC B.32]

$$\exists v0. \left(\begin{array}{l} (Lift(s))(\alpha,\alpha 0) \\ \wedge \; (l \rightarrow Skip)(\alpha 0,\alpha') \end{array} \right) ; A$$

= [Sequence Denotational Definition B.33]

Lift (s) ; l $\rightarrow$ Skip

□

Theorem 8. ***Parallel Prefixed Loc**:*

(loc s1 • A1) $[\![$ *x1 | CS | x2* $]\!]$ *(loc s2 • A2)*

$\sqsubseteq$

l $\rightarrow$ *((loc s3 • A3)* $[\![$ *x1 | CS | x2* $]\!]$ *(loc s2 • A2))*

provided that

Lift (s1) ; A1 $\sqsubseteq$ *Lift (s3) ; l* $\rightarrow$ *A3* □ *Lift (s1) ; A1* ***[Assumption 1]***

(assumption 1 is the right side of the denotational definition of labelled transition

(c1 | s1 $\models$ *A1)* $\xrightarrow{l}$ *(c3 | s3* $\models$ *A3))*

initials (Lift (s2) ; A2) $\subseteq$ *CS* ***[Assumption 2]***

(assumption 2 indicates that we require that the initial events being offered by Lift (s2) ; A2 lie on channel set CS)

CS $\cap$ *usedC (Lift (s3); A3)* = $\emptyset$ ***[Assumption 3]***

(assumption 3 demands that no channel used on Lift (s3); A3 can lie on channel set CS)

wrtV (Lift (s3); A3) $\cap$ *usedV (Lift (s2) ; A2)* = $\emptyset$ ***[Assumption 4]***

(assumption 4 demands that no variables that are used by action Lift (s2) ; A2 can be written)

Lift (s2) ; A2 is divergence free ***[Assumption 5]***

(assumption 5 is self-explanatory)

ok = ok' ***[Assumption 6]***

(assumption 6 demands that either (1) if the program has started than it has finished, or (2) if the program did not start then it did not finish)

Proof

Before starting to prove, we will derive a new assumption from the provided assumption:
[Assumption 1] Lift (s1); A1 $\sqsubseteq$ Lift (s3); l $\rightarrow$ A3 $\Box$ Lift (s1); A1
$\Rightarrow$ **[External Choice Refinement Implication 15]**
[Assumption ppl] Lift (s1) ; A1 $\sqsubseteq$ Lift (s3) ; l $\rightarrow$ A3

We also assume Theorem 11 on the proof to assure that Lift (s1), Lift (s2) and Lift (s3) are *Circus* actions. We will add an explanation of each tactic and sub-goal, envisaging explaining this proof. Now we will start proving:
((loc s1 • A1) $[\![$ x1 | CS | x2 $]\!]$ (loc s2 • A2))
= [Definition of loc B.29]
(We will apply the definitions of loc in order to convert them to Lift expressions. Lift expressions allow the application of the lemmas and theorems used throughtout the proof)
((Lift (s1) ; A1) $[\![$ x1 | CS | x2 $]\!]$ (Lift (s2) ; A2))
$\sqsubseteq$ **[Assumption ppl with Parallelism Refinement Monotonic B.2.21]**
(application of assumption ppl and Monotonicity of Refinement for Parallelism allow refining the expression to an expression that has l $\rightarrow$ A3, linking the left expression to the right expression)
((Lift (s3) ; l $\rightarrow$ A3) $[\![$ x1 | CS | x2 $]\!]$ (Lift (s2) ; A2))
= [Lift Shift (having Assumption 6) 7]
(Lift Shift has an important role on this proof, as it allows swapping the positions from Lift (s3) and l on the prefixing ($\rightarrow$) operator. This is important because the right expression of the refinement we want to prove has l before expression loc s3 • A3)
((l $\rightarrow$ Lift (s3) ; A3) $[\![$ x1 | CS | x2 $]\!]$ (Lift (s2) ; A2))
= [Prefixing Sequence theorem: B.2.5]
(now we transform l $\rightarrow$ Lift (s3) into l $\rightarrow$ Skip ; Lift (s3), in order to allow the application of Parallel Composition Sequence Step)
((l $\rightarrow$ Skip ; Lift (s3) ; A3) $[\![$ x1 | CS | x2 $]\!]$ (Lift (s2) ; A2))
= [Parallel Composition Sequence Step (having assumptions 2 to 5) B.2.3]
(the role of this step is to take l outside the left branch, making it a prefix not only from the left branch of the parallel composition, but also from the whole parallel composition)
l $\rightarrow$ ((Lift (s3) ; A3) $[\![$ x1 | CS | x2 $]\!]$ (Lift (s2) ; A2))
= [Definition of loc B.29]
(now we apply again the definition of loc in order to reach the right expression of the refinement)
l $\rightarrow$ ((loc s3 • A3) $[\![$ x1 | CS | x2 $]\!]$ (loc s2 • A2))

□

Theorem 9. *Parallel Independent Refinement:*
(loc s1 • A1) $[\![x1 \mid CS \mid x2]\!]$ *(loc s2 • A2)* $\sqsubseteq$ *(loc s3 • A3)* $[\![x1 \mid CS \mid x2]\!]$ *(loc s2 • A2)*

provided that

- *Lift (s1) ; A1* $\sqsubseteq$ *Lift (s3) ; A3*

In order to prove, we use Theorem 11 to assume that all Lift (s_i) are *Circus* actions.

Proof

$$\left(\begin{array}{l} (locs1 \bullet A1) \,[\![x1 \mid CS \mid x2]\!]\, (locs2 \bullet A2) \\ \sqsubseteq (locs3 \bullet A3) \,[\![x1 \mid CS \mid x2]\!]\, (locs2 \bullet A2) \end{array} \right)$$

= [Definition of loc B.29]

$$\left(\begin{array}{l} (Lift(s1);\, A1) \,[\![x1 \mid CS \mid x2]\!]\, (Lift(s2);\, A2) \\ \sqsubseteq (Lift(s3);\, A3) \,[\![x1 \mid CS \mid x2]\!]\, (Lift(s2);\, A2) \end{array} \right)$$

= [Assumption, Ref. Eq. Sides 1 and Refinement Monotonic 6]

true

□

Theorem 10. *Lifted Assignment is a Circus Assignment Command:*
Lift (x := c) = (x := c)

Proof

Lift (x := c)

= [Definition of Lift B.24]

R1 ∘ R3 (true ⊢ x := c ∧ tr' = tr ∧ ¬ wait')

= [Replacing tr' = tr by tr' - tr = $\langle\,\rangle$**]**

R1 ∘ R3 (true ⊢ x := c ∧ tr' - tr = $\langle\,\rangle$ ∧ ¬ wait')

= [The expression is also R2. If it is R1 and R3, it is R]

R (true ⊢ x := c ∧ tr' - tr = $\langle\,\rangle$ ∧ ¬ wait')

= [State Assignment B.16]

R (true ⊢ x' = c ∧ u' = u ∧ tr' - tr = $\langle\,\rangle$ ∧ ¬ wait')

= [Assignment Denotational Definition A.4]

x := c □

Theorem 11. *Lifted Sequence of Assignments is a Circus Assignment Command:*
Lift (s) = s

provided that

- s is a sequence of state assignments (B.16) of the type ($x_1 := c_1$; ...; $x_n := c_n$);

Proof
Lift (s)
= **[Assumption]**
Lift ($x_1 := c_1$; ...; $x_n := c_n$)
= **[Lift Composition B.2.27]**
Lift ($x_1 := c_1$); ...; Lift ($x_n := c_n$)
= **[Lifted Assignment is a Circus Assignment Command 10 (n times)]**
$x_1 := c_1$; ...; $x_n := c_n$
= **[Assumption]**
s
□

Theorem 12. **CSP process as a Self-Reactive Design:*
$A = R((\neg\ A^f_f) \vdash (A^t_f))$, provided that A is a CSP process (CSP1, CSP2 and CSP3-healthy)

This theorem lies on [6].

Theorem 13. ****Circus Action CSP1-CSP2-CSP3-Healthy:***
Every Circus action is CSP1 B.38, CSP2 B.39 and CSP3 B.40 Healthy.

This theorem lies on [6].

Theorem 14.

****Monotonicity of Refinement on External Choice**:*
$A_1 \Box A_2 \sqsubseteq B_1 \Box B_2$ provided $A_1 \sqsubseteq B_1$ and $A_2 \sqsubseteq B_2$

Proof The proof lies on the document [7].
□

Theorem 15.

External Choice Refinement Implication
((Lift (s1) ; A1) $\sqsubseteq$ (Lift (s2) ; $l \rightarrow$ A2) $\Box$ (Lift (s1) ; A1))
$\Rightarrow$
((Lift (s1) ; A1) $\sqsubseteq$ (Lift (s2) ; $l \rightarrow$ A2))

provided that A1 = l → A2

Proof
((Lift (s1) ; A1) ⊑ (Lift (s2) ; l → A2) □ (Lift (s1) ; A1))
[Assumption]
((Lift (s1) ; A1) ⊑ (Lift (s2) ; A1) □ (Lift (s1) ; A1))
⇒ **[Refinement Implication For Subst.Equiv. B.4 (having Lift Subst. Equiv. B.5 both for Lift (s1) and Lift (s2)) + Monotonicity of Refinement for Ext.Choice 14)]**
((Lift (s1) ; A1) ⊑ (Lift (s2) ; A1)
= [Assumption]
((Lift (s1) ; A1) ⊑ (Lift (s2) ; l → A2)
□

Theorem 16. ***Hidden Event Sequenced by Skip****:*
((l → Skip) \ {l}) = Skip

Proof
The proof lies on [6]. □

B.2.4 Auxiliary Laws

On this appendix, we will list all denotational laws that were used throughtout this document (all refinement laws that were used from the Denotational Semantics of *Circus* [6] (refinement laws), laws concerning the denotational theory of the CML Operational Semantics we have lifted, based on [3], and laws that we created and proved correct).

Refinement Laws

Law B.2.1. *(*) Skip = V : [g, true]*

The above law is shown on [6].

Law B.2.2. *(*) Skip ; A = A*

Proof The proof lies on the document [6].
□

Law B.2.3. ***Parallel Composition Sequence Step**:*
$(A1 \ ; \ A2) \ [\![\ ns1 \ \{\!| \ cs \ |\!\} \ ns2 \ ]\!] \ A3 = A1 \ ; \ (A2 \ [\![\ ns1 \ \{\!| \ cs \ |\!\} \ ns2 \ ]\!] \ A3)$

provided that

- $initials \ (A3) \subseteq cs$
- $cs \cap usedC \ (A1) = \emptyset$
- $wrtV \ (A1) \cap usedV \ (A3) = \emptyset$
- *A3 is divergence-free*

Function usedV (Act) is defined at [6] and returns the set of variables that are used by action Act. Function usedC is defined on B.12. Function wrtV (Act) is defined at [6] and returns the set of variables that are written by action Act.

Proof The proof lies on the document [6].

□

Law B.2.4. ***True Guard**: $true \ \& \ A = A$*

Proof The proof lies on the document [6]

□

Law B.2.5. ***Prefixing Sequence**: $a \rightarrow B = a \rightarrow Skip \ ; \ B$*

The proof lies on the document [6]

Law B.2.6.

***Compound Actions Commutative (Except for Sequence)**:*
$A1 \ BOP \ A2 = A2 \ BOP \ A1$
provided that $BOP \in \{ \ [\![\ \{\!| \ cs \ |\!\} \ ]\!], \ |||, \ \Box, \ \sqcap, \ \}$

For Parallel Compositions with name-sets, the name-sets are also commuted:
$A1 \ [\![\ ns1 \ \{\!| \ cs \ |\!\} \ ns2 \ ]\!] \ A2 = A2 \ [\![\ ns2 \ \{\!| \ cs \ |\!\} \ ns1 \ ]\!] \ A1$

Proof There is a version of this law for each compound operator that is commutative, all of which are referenced on document [6]..

□

Law B.2.7.

***Communication Parallelism Distribution**:*
$c \rightarrow (A1 \ [\![\{\!| CS |\!\}]\!] \ A2) = ((c \rightarrow A1) \ [\![\{\!| CS |\!\}]\!] \ (c \rightarrow A2))$
provided that $\{ c \} \in CS$

Proof The law is referenced on document [6].

□

Law B.2.8. ***Hiding Identity**:*
$A \setminus CS = A$
provided $CS \cap UsedC\ (A) = \emptyset$

Proof The law is referenced on document [6]. □

Law B.2.9. ***Hiding Monotonic**:*
$P_1 \setminus S \sqsubseteq P_2 \setminus S$
provided $P_1 \sqsubseteq P_2$

Proof The law is referenced on document [6]. □

Law B.2.10. ***Hiding External Choice Distributive**:*
$(A_1 \ \Box \ A_2) \setminus cs = (A_1 \setminus cs) \ \Box \ (A_2 \setminus cs)$
provided $(initials\ (A1) \cup initials\ (A2)) \cap cs = \emptyset$

Proof The law is referenced on document [6]. □

Law B.2.11. ***External Choice Sequence Distributive**:*
$((A_1 \ ; B) \ \Box \ (A_2 \ ; B)) = ((A_1 \ \Box \ A_2) \ ; B))$

Proof The law is referenced on document [6]. □

Law B.2.12. ***Hiding Sequence Distributive**:*
$(A_1 \ ; A_2) \setminus cs = (A_1 \setminus cs) \ ; (A_2 \setminus cs)$

Proof The law is referenced on document [6]. □

Law B.2.13. ***Variable Block Extension**:*
$A_1 \ ; (\mathbf{var}\ x : T \bullet A_2) \ ; A_3 = (\mathbf{var}\ x : T \bullet A_1 \ ; A_2 \ ; A_3)$
provided $x \notin FV\ (A_1) \cup FV\ (A_3)$

Proof The law is referenced on document [6]. □

Law B.2.14. ***Variable Block Extension (A):***
A_1 ; (**var** $x : T \bullet A_2$) = (**var** $x : T \bullet A_1$; A_2)
provided $x \notin FV(A_1)$

Proof The law is referenced on document [6]. □

Law B.2.15. ***Circus Basic Process to Circus action:***
begin state *[decl | pred] PPars* • *A* **end** $\widehat{=}$ **var** *decl* • *A*

Proof The law is referenced on document [6]. □

Lemma B.18. *(*) STOP* $\Box$ *A = A*

Proof The proof lies on [6]. □

Law B.2.16. ***Sequence, External and Internal Choice Process to Basic Process:***
For op $\in \{ ; , \Box, \sqcap \}$
P op Q = **begin state** *State* $\widehat{=}$
P.State $\wedge$ *Q.State*
P.PPar $\wedge_{\Xi}$ *Q.State*
Q.PPar $\wedge_{\Xi}$ *P.State*
• *P.Act op Q.Act*
end

Proof The law is referenced on document [6]. □

Law B.2.17. ***Parallel and Interleave Processes to Basic Process:***
For op = $[\![CS]\!]$
P $[\![cs]\!]$ *Q* = **begin state** *State* $\widehat{=}$
P.State $\wedge$ *Q.State*
P.PPar $\wedge_{\Xi}$ *Q.State*
Q.PPar $\wedge_{\Xi}$ *P.State*
• *P.Act* $[\![\alpha(P.State) \mid cs \mid \alpha(Q.State)]\!]$ *Q.Act*
end

Proof The law is referenced on document [6]. □

Law B.2.18. ***Rename Basic Process to Basic Process with Rename Main Action:***
*(***begin** *PARS* • *A* **end***) [a1, ... an := b1, ..., bn]*
*= (***begin** *PARS* • *A [a1, ... an := b1, ..., bn]* **end***)*

Proof The proof lies on [6] □

Law B.2.19. *(*)* **begin** *PPars* • *A* **end** \ *CS* = **begin** *PPars* • *(A* \ *CS)* **end**

The law is referenced on document [6].

Law B.2.20.

***Internal Choice Refinement Monotonic**:*
$A_1 \sqcap A_2 \sqsubseteq B_1 \sqcap B_2$ *provided* $A_1 \sqsubseteq B_1$ *and* $A_2 \sqsubseteq B_2$

Proof Proof lies on document [6]. □

Law B.2.21.

***Parallelism Refinement Monotonic**:*
$A_1 \ [\![CS]\!] \ A_2 \sqsubseteq B_1 \ [\![CS]\!] \ B_2$ *provided* $A_1 \sqsubseteq B_1$ *and* $A_2 \sqsubseteq B_2$

Proof Proof lies on document [6]. □

Law B.2.22.

***External Choice Refinement Monotonic**:*
$A_1 \ \Box \ A_2 \sqsubseteq B_1 \ \Box \ B_2$ *provided* $A_1 \sqsubseteq B_1$ *and* $A_2 \sqsubseteq B_2$

Proof Proof lies on document [6]. □

Law B.2.23. ***R2 Conjunction Not Mentioning Trace*** $p \wedge R2\ (P) = R2\ (p \wedge P)$, *provided that p does not mention tr and tr'*

The proof lies on the documents [6].

Cavalcanti and Woodcock's Laws

Law B.2.24. ***External Choice Idempotence**:*
$(P \ \Box \ P) = P$

Proof The proof lies on the document [3]

□

Law B.2.25. ***Refinement of Non-Determinism**:*
$A \sqcap B \sqsubseteq A$

Proof The proof lies on document [3]

□

Law B.2.26. ***Lift is CSP4**: Lift (v := e) ; Skip = Lift (v := e)*

Proof The proof lies on the document [3]. All definitions, lemmas, theorems, laws and rules from CML can be used for *Circus* [5].
□

Law B.2.27. ***Lift Composition**: Lift (s ; v := w_1) = Lift (s) ; Lift (v := w_1)*

Proof The proof lies on the document [3]
□

Law B.2.28. ***Lift External Choice**: Lift(s) ; (P □ Q) = (Lift(s) ; P) □ (Lift(s) ; Q)*

Proof The proof lies on the document [3]
□

Law B.2.29.

***Lift Left Unit**: Lift (s) ; Lift (t) = Lift (t)*

Proof The proof lies on the document [3]
□

Law B.2.30. ***Lift Leading Substitution**:*
Lift (s) ; P = Lift (s) ; P [e / w_1], provided that (s ; (w_1 = e))

Proof The proof lies on the document [3]
□

Law B.2.31. *(*) Lift (s) ; Lift (v := e) = Lift (s) ; v :=$_{RD}$ e*

Proof The proof lies on the document [3]
□

Law B.2.32.

***Substitution**:*
Lift (v := w) [e / w] = Lift (v := e)

Proof The proof lies on the document [3]

□

Law B.2.33. *(*) Lift (var x := w_1) = var x ; Lift (x := w_1)*

Proof The proof lies on document [3]

□

Law B.2.34. ****Reactive Design Assignment Declaration**:*
var x ; x :=$_{RD}$ w = var x :=$_{RD}$ w

Proof The proof lies on document [3]

□

Law B.2.35. *(*) var$_{RD}$ x :=$_{RD}$ w ; P = P (w ∨ x)*

Proof The proof lies on document [3]

□

Law B.2.36. ****Input Absorption**:*
Lift(s) ; d?x : T → A = (Lift(s) ; d.w → A [w / x]) □ (Lift(s) ; d?x : T → A)

Proof The proof lies on document [3]
□

Law B.2.37. ****Parallel Distributivity**:*
Lift (s) ; (P ‖[x_1 | cs | x_2]‖ Q) = (Lift(s) ; P) ‖[x_1 | cs | x_2]‖ (Lift(s) ; Q)

Proof The proof lies on document [3]
□

Law B.2.38. ****Lift Semi-Idempotence**:*
s ; (s | x_1)$_{+x_2}$ = s

Proof The proof lies on document [3]
□

Law B.2.39. ****Lift Merge**:*
Lift (s_1) ‖[x_1 ∣ cs ∣ x_2]‖ Lift (s_2) = Lift ((s_1 ∣ x_1) ∧ (s_2 ∣ x_2))

Proof The law is referenced on document [3] □

Law B.2.40. ***Closure Conjunctive R1** *R1 (P ∧ Q) = P ∧ Q, provided that P and Q are R1*

Law B.2.41. ***Closure Conjunctive R3** *R3 (P ∧ Q) = P ∧ Q, provided that P and Q are R3*

This law was proved on [2].

Law B.2.42. ***Closure Conjunctive R** *R (P ∧ Q) = P ∧ Q, provided that P and Q are R*

This law was proved on [2].

Law B.2.43. ***Closure Disjunctive R2** *R2 (P ∨ Q) = P ∨ Q, provided that P and Q are R2*

This law was proved on [2].

Other Laws

On this sub-section we show laws that we created and proved correct.

Law B.2.44. ***Parallel Composition Prefixing Step**:*

(a → A2) [| ns1 {| cs |} ns2 |] A3 = a → (A2 [| ns1 {| cs |} ns2 |] A3)

provided that

initials (A3) ⊆ cs
cs ∩ {a} = ∅
wrtV ({a}) ∩ usedV (A3) = ∅
A3 is divergence-free

Proof

(a → A2) [| ns1 {| cs |} ns2 |] A3
= [Prefixing Sequence theorem: B.2.5]
((a → Skip) ; A2) [| ns1 {| cs |} ns2 |] A3
= [Parallel Composition Sequence Step B.2.3 and Assumptions]
true
□

Law B.2.45. ***Lifted Assignment Sequenced by end x***
Lift (s ; end x) = Lift (s) ; Lift (end x)

Proof
Lift (s ; end x)
= [Definition of Lift B.24]
R1 ∘ R3 (true ⊢ s ; end x ∧ tr' = tr ∧ ¬ wait')
= [Definition of Sequence B.33]
R1 ∘ R3 (true ⊢ (∃ v0 . s (α, α 0) ∧ end x (α 0, α ')) ∧ tr' = tr ∧ ¬ wait')
= [Predicate Calculus: all variables on v0 are free on R1 and R3]
∃ v0 . R1 ∘ R3 (true ⊢ (s (α, α0) ∧ end x (α0, α')) ∧ tr' = tr ∧ ¬ wait')
= [Predicate Calculus]
∃v0 . R1 ∘ R3 (true ⊢ s (α, α0) ∧ tr' = tr ∧ ¬ wait' ∧ end x (α0, α') ∧ tr' = tr ∧ ¬ wait')
= [Designs And True B.2.46]
∃ v0 .
R1 ∘ R3 (true ⊢ s (α, α0) ∧ tr' = tr ∧ ¬ wait')
∧ R1 ∘ R3 (true ⊢ end x (α0, α') ∧ tr' = tr ∧ ¬ wait')
= [Definition of Lift B.24]
∃ v0 . Lift (s) ∧ Lift (end x)
= [Definition of Sequence B.33, having, from theorem 11 that Lift (s) is a Circus action]
Lift (s) ; Lift (end x)
□

Law B.2.46. ***Designs And True:***
(true ⊢ Q_1 ∧ Q_2) = (true ⊢ Q_1) ∧ (true ⊢ Q_2)

Proof true ⊢ Q1 ∧ Q2
= [Predicate Calculus]
true ∧ ok ⇒ (Q1 ∧ Q2 ∧ ok')
= [Predicate Calculus]
ok ⇒ (Q1 ∧ Q2 ∧ ok')
= [Predicate Calculus]
(¬ok) ∨ (Q1 ∧ Q2 ∧ ok')
= [Predicate Calculus]
(¬ok ∨ Q1) ∧ (¬ok ∨ Q2) ∧ (¬ok ∨ ok')
= [Predicate Calculus]
((¬ok ∨ Q1) ∧ (¬ok ∨ ok')) ∧ ((¬ok ∨ Q2) ∧ (¬ok ∨ ok'))

= [Predicate Calculus]
(¬ok ∨ (Q1 ∧ ok')) ∧ (¬ok ∨ (Q2 ∧ ok'))
= [Predicate Calculus]
(ok ⇒ Q1 ∧ ok') ∧ (ok ⇒ Q2 ∧ ok')
= [Predicate Calculus]
(true ∧ ok ⇒ Q1 ∧ ok') ∧ (true ∧ ok ⇒ Q2 ∧ ok')
= [Predicate Calculus]
(true ⊢ Q1) ∧ (true ⊢ Q2)
□

Law B.2.47. *P ◁ b ▷ (Q ∧ R) = (P ◁ b ▷ Q) ∧ (P ◁ b ▷ R)*

Proof P ◁ b ▷ (Q ∧ R)
= [Definition of Conditional]
(b ∧ P) ∨ (¬ b ∧ Q ∧ R)
= [Predicate Calculus]
((b ∧ P) ∨ (¬ b ∧ Q ∧ ¬ b ∧ R))
= [Predicate Calculus]
((b ∧ P) ∨ (¬ b ∧ Q)) ∧ ((b ∧ P) ∨ (¬ b ∧ R))
= [Definition of Conditional]
(P ◁ b ▷ Q) ∧ (P ◁ b ▷ R)
□

Law B.2.48. ***Lift And**:*
Lift (S ∧ P) = Lift (S) ∧ Lift (P)

Proof LHS
= Lift (S ∧ P)
= [Definition of Lift (B.24)]
R1 (R3 (true ⊢ S ∧ P ∧ tr' = tr ∧ ¬ wait'))
= [Predicate Calculus]
R1 (R3 (true ⊢ S ∧ tr' = tr ∧ ¬ wait' ∧ P ∧ tr' = tr ∧ ¬ wait'))
= [Predicate Calculus]
R1 (R3 (true ⊢ (S ∧ tr' = tr ∧ ¬ wait') ∧ (P ∧ tr' = tr ∧ ¬ wait')))
= [Designs And True B.2.46]
R1 ∘ R3 ((true ⊢ (S ∧ tr' = tr ∧ ¬ wait')) ∧ (true ⊢ (P ∧ tr' = tr ∧ ¬ wait')))
= [Closure Conjunctive R1 (B.2.40) and Closure Conjunctive R3 (B.2.41)]
R1 ∘ R3 (true ⊢ (S ∧ tr' = tr ∧ ¬ wait')) ∧ R1 ∘ R3 (true ⊢ (P ∧ tr' = tr ∧ ¬ wait'))

= [Definition of Lift (B.24)]
Lift (S) $\wedge$ Lift (P)
= RHS □

Law B.2.49. ***R2 and R3 Composition Commutative:***
R2 $\circ$ R3 (P) = R3 $\circ$ R2 (P)

Proof
R2 $\circ$ R3 (P)
= [Definition of R3 B.21]
R2 ($II_{rea} \lhd$ wait $\rhd$ P)
= [Definition of Conditional B.18]
R2 ((wait $\wedge$ II_{rea}) $\vee$ ($\neg$ wait) $\wedge$ P))
= [Disjunctive Closure R2 B.2.43]
R2 (wait $\wedge$ II_{rea}) $\vee$ R2 (($\neg$ wait) $\wedge$ P)
= [R2 Conjunctive Not Mentioning Trace B.2.23 (having that wait and $\neg$ wait do not mention tr and tr')]
(wait $\wedge$ R2 (II_{rea})) $\vee$ (($\neg$ wait) $\wedge$ R2 (P))
= [Definition of Conditional B.18]
(R2 (II_{rea})) $\lhd$ wait $\rhd$ (R2 (P))
= [Definition of R3 B.21]
R3 $\circ$ R2 (P)
□

B.3 Proof of Soundness for rules

B.3.1 Assignment

Attached Rule 1.

Assignment:

$$\frac{c \qquad (s\ ;\ (w0 = e))}{(c \mid s \models v := e) \xrightarrow{\tau} (c \wedge (s\ ;\ (w0 = e)) \mid s\ ;\ v := w0 \models Skip)}$$

The proof, on document [3], consists on proving that the LHS of the refinement expression equals the RHS.

Proof
RHS
=
Lift (s ; v := w0) ; Skip
= **[Law LiftCSP4 B.2.26]**
Lift (s ; v := w0)
= **[Law Lift Composition B.2.27]**
Lift (s) ; Lift (v := w0)
= **[Law Lift Leading Substitution** B.2.30]
Lift (s) ; Lift (v := w0) [e/w0]
= **[Law Substitution B.2.32]**
Lift (s) ; Lift (v := e)
= **[Reactive Design Assignment B.17]**
Lift (s) ; v :=$_{RD}$ e
=
LHS □

B.3.2 Prefixing*

Attached Rule 2.

Input*:

$$\frac{c \quad T \neq \emptyset \quad x \notin \alpha(s)}{(c \mid s \models d?x{:}T \to A) \xrightarrow{d.w_0} (c \wedge w_0 \in T \mid s \,;\, \textbf{var}\ x := w_0 \models \textit{let}\ x \bullet A)}$$

Proof

(c | s $\models$ d?x:T $\to$ A) $\xrightarrow{d.w_0}$ (c $\wedge$ w_0 $\in$ T | s ; **var** x := w_0 $\models$ **let** x $\bullet$ A)

= **[Definition B.26]**

$\forall$ w . c $\wedge$ w_0 : T $\Rightarrow$

$\big($ Lift(s); d?x: T $\to$ A $\sqsubseteq$ (Lift(s; **var** x := w_0); $d._1$ $\to$ **let** x $\bullet$ A) $\Box$ (Lift(s); d?x: T $\to$ A) $\big)$

On Deliverable [3], the proof consists on proving that the RHS (Right-Hand Side) equals the LHS (Left-Hand Side):

RHS

=

(Lift(s; **var** x := w_0); $d._1$ $\to$ **let** x $\bullet$ A) $\Box$ (Lift(s); d?x: T $\to$ A)

= **[Lemma B.28]**

(Lift (s ; **var** x := w_0) ; d.w_0 $\to$ A) $\Box$ (Lift(s) ; d?x : T $\to$ A)

= **[Law Lift Composition B.2.27]**

(Lift(s); Lift(**var** x := w_0); d.w_0 $\to$ A) $\Box$ (Lift(s); d?x : T $\to$ A)

= **[Law Lift Var B.2.33]**

$\Big($ $(Lift(s);$ **var**$x;$ $Lift(x := w_0);$ $d.w_0 \to A)$ $\Box$ $(Lift(s);$ $d?x : T \to A)$ $\Big)$

= **[Reactive Design Assignment B.17]**

(Lift(s); **var** x; x :=$_{RD}$ w_0; d.w_0 $\to$ A) $\Box$ (Lift(s); d?x : T $\to$ A)

= **[Reactive Design Declaration B.2.34]**

(Lift(s); **var** x :=$_{RD}$ w_0; d.w_0 $\to$ A) $\Box$ (Lift(s); d?x : T $\to$ A)

= **[Reactive Design Declaration Elimination B.2.35]**

(Lift(s); (d.w_0 $\to$ A) (w_0/x)) $\Box$ (Lift(s) ; d?x : T $\to$ A)

= **[Substitution B.2.32]**

(Lift(s); d.w_0 $\to$ A(w_0/x)) $\Box$ (Lift(s); d?x : T $\to$ A)

= **[Input Absorption B.2.36]**

(Lift(s) ; d?x : T $\to$ A)

=
LHS □

Attached Rule 3.

Output**:

$$\frac{c \quad s\ ;\ (w_0 = e)}{(c \mid s \models d!e \to A) \xrightarrow{d.w_0} (c \wedge s\ ;\ (w_0 = e) \mid s \models A)}$$

Proof

$(c \mid s \models d!e \to A) \xrightarrow{d.w_0} (c \wedge s\ ;\ (w_0 = e) \mid s \models A)$

= **[Definition B.26]**

$\forall\, w \cdot c \wedge (c \wedge s\ ;\ (w_0 = e)) \Rightarrow$

$\big(\text{Lift}\ (s)\ ;\ d.e \to A \sqsubseteq (\text{Lift}\ (s)\ ;\ d.w_0 \to A) \,\square\, (\text{Lift}\ (s)\ ;\ d.e \to A)\big)$

= **[Lift Leading Substitution B.2.30] and assumption (s ; (w_0 = e))**

$\forall\, w \cdot c \wedge (c \wedge s\ ;\ (w_0 = e)) \Rightarrow$

$\big(\text{Lift}\ (s)\ ;\ d.e \to A \sqsubseteq (\text{Lift}\ (s)\ ;\ (d.w_0 \to A)\ [e\ /\ w_0]) \,\square\, (\text{Lift}\ (s)\ ;\ d.e \to A)\big)$

= **[Substitution B.2.32] and assumption (s ; (w_0 = e))**

$\forall\, w \cdot c \wedge (c \wedge s\ ;\ (w_0 = e)) \Rightarrow$

$\big(\text{Lift}\ (s)\ ;\ d.e \to A \sqsubseteq (\text{Lift}\ (s)\ ;\ d.e \to A\ [e\ /\ w_0]) \,\square\, (\text{Lift}\ (s)\ ;\ d.e \to A)\big)$

= **[Substitution B.2.32] and w_0 does not occur in A**

$\forall\, w \cdot c \wedge (c \wedge s\ ;\ (w_0 = e)) \Rightarrow$

$\big(\text{Lift}\ (s)\ ;\ d.e \to A \sqsubseteq (\text{Lift}\ (s)\ ;\ d.e \to A) \,\square\, (\text{Lift}\ (s)\ ;\ d.e \to A)\big)$

= **[External Choice Idempotence B.2.24]**

$\forall\, w \cdot c \wedge (c \wedge s\ ;\ (w_0 = e)) \Rightarrow \big(\text{Lift}\ (s)\ ;\ d.e \to A \sqsubseteq \text{Lift}\ (s)\ ;\ d.e \to A\big)$

= **[Refinement Equal Sides 1]**

$\forall\, w \cdot c \wedge (c \wedge s\ ;\ (w_0 = e)) \Rightarrow \text{true}$

= **[Predicate Calculus]**

$\forall\, w \cdot \text{true}$

= **true** □

B.3.3 Variable Block

All laws for Variable Block (Variable Block Begin, Variable Block Visible and Variable Block End) were proved by Barrocas, on this Thesis.

Attached Rule 4.

Variable Block Begin:

$(c \mid s \models \textbf{var}\ x : T \bullet A) \xrightarrow{\tau} (c \wedge w_0 \in T \mid s\ ;\ \textbf{var}\ x := w_0 \models \textbf{let}\ x \bullet A)$

Proof

(c ∣ s ⊨ **var** x : T • A) $\xrightarrow{\tau}$ (c ∧ w_0 ∈ T ∣ s ; **var** x := w_0 ⊨ **let** x • A)

= [Definition of Silent Transition B.25]

∀w . (c ∧ c ∧ w_0 : T) ⇒ (Lift (s) ; **var** x : T • A) ⊑ (Lift (s ; **var** x := w_0) ; let x • A)

= [w is universally quantified, so we abstract it]

(c ∧ c ∧ w_0 : T) ⇒ (Lift (s) ; **var** x : T • A) ⊑ (Lift (s ; **var** x := w_0) ; let x • A)

= [Lift Composition B.2.27]

(c ∧ c ∧ w_0 : T) ⇒ (Lift (s) ; **var** x : T • A) ⊑ (Lift (s) ; Lift (**var** x := w_0) ; let x • A)

= [Lift Left Unit B.2.29]

(c ∧ c ∧ w_0 : T) ⇒ (**var** x : T • A) ⊑ (Lift (**var** x := w_0) ; let x • A)

= [(x := e) = (var x ; x := e)]

(c ∧ c ∧ w_0 : T) ⇒ (**var** x : T • A) ⊑ (Lift (**var** x; x := w_0) ; let x • A)

= [Let definition B.28]

(c ∧ c ∧ w_0 : T) ⇒ (**var** x : T • A) ⊑ (Lift (**var** x; x := w_0) ; A)

= [Refinement Definition B.14]

(c ∧ c ∧ w_0 : T) ⇒ [(Lift (**var** x; x := w_0) ; A) ⇒ (**var** x : T • A)]

= [Lemma B.8 and assms (w_0 : T)]

(c ∧ c ∧ w_0 : T) ⇒ [true]

= [Predicate Calculus]

(c ∧ c ∧ w_0 : T) ⇒ true

= [Predicate Calculus]

true

□

Attached Rule 5.

Variable Block Visible:

$$\frac{(c_1 \mid s_1 \models A_1\ ) \xrightarrow{l} (c_2 \mid s_2 \models A_2)}{(c_1 \mid s_1 \models \textbf{let}\ x \bullet A_1\ ) \xrightarrow{l} (c_2 \mid s_2 \models \textbf{let}\ x \bullet A_2)}$$

Proof

$(c_1 \mid s_1 \models \textbf{let}\ x \bullet A_1) \xrightarrow{l} (c_2 \mid s_2 \models \textbf{let}\ x \bullet A_2)$

= [Definition of let B.28]

$(c_1 \mid s_1 \models A_1) \xrightarrow{l} (c_2 \mid s_2 \models \textbf{let}\ x \bullet A_2)$

= [Definition of let B.28]

$(c_1 \mid s_1 \models A_1) \xrightarrow{l} (c_2 \mid s_2 \models A_2)$

= [Assumption $(\mathbf{c_1 \mid s_1 \models A_1}) \xrightarrow{l} (\mathbf{c_2 \mid s_2 \models A_2})$]

true

□

Attached Rule 6.

***Variable Block End**:*

$$\frac{c}{(c \mid s \models \textbf{\textit{let}}\ x \bullet Skip) \xrightarrow{l} (c \mid s\ ;\ \textbf{\textit{end}}\ x \models Skip)}$$

Proof

$\forall w . (c \wedge c) \Rightarrow (Lift(s);\ let\,x \bullet Skip) \sqsubseteq (Lift(s;\ end\,x);\ Skip)$

= [w is universally quantified, so we abstract it]

$(c \wedge c) \Rightarrow (Lift(s);\ let\,x \bullet Skip) \sqsubseteq (Lift(s;\ end\,x);\ Skip)$

= [Assumption c]

$(Lift(s);\ let\,x \bullet Skip) \sqsubseteq (Lift(s;\ end\,x);\ Skip)$

= [Definition of let [B.28]]

$(Lift(s);\ Skip) \sqsubseteq (Lift(s;\ end\,x);\ Skip)$

= [Lifted Assignment Sequenced by end x B.2.45]

$(Lift(s);\ Skip) \sqsubseteq (Lift(s);\ Lift(end\,x);\ Skip)$

As Sequence is monotonic with respect to refinement and Lift (s) is refined by itself, if we prove that Skip ⊑ (Lift (end x) ; Skip), we also prove the above expression:

$Skip \sqsubseteq (Lift(end\,x);\ Skip)$

= [Sequence Denotational Definition B.33]

$Skip \sqsubseteq \exists\, v0 . (Lift(end\,x)[v, v0] \wedge Skip[v0, v'])$

= [end x definition]

$Skip \sqsubseteq \exists v0.(Lift(true \wedge \exists x')[v, v0] \wedge Skip[v0, v'])$

= [Lift Definition B.24 and true ∧ ∃ x' is R2]

$Skip \sqsubseteq \exists v0.(R(true \vdash true \wedge \exists x0.tr0 = tr \wedge \neg\, wait0) \wedge Skip[v0, v'])$

= [Skip Denotational Definition]

$Skip \sqsubseteq$
$\exists v0. \left(\begin{array}{l} R(true \vdash true \wedge \exists x0.tr0 = tr \wedge \neg\, wait0) \\ \wedge\, (R(true \vdash tr' = tr0 \wedge \neg\, wait' \wedge v' = v0)) \end{array} \right)$
= [Predicate Calculus]
$Skip \sqsubseteq$
$\exists\, v0\,. \left(\begin{array}{l} R\,(true \vdash true \wedge tr0 = tr \wedge \neg\, wait0) \\ \wedge\, (R(true \vdash tr' = tr0 \wedge \neg\, wait' \wedge v' = v0)) \end{array} \right)$
= [Skip definition]
$R(true \vdash tr' = tr \wedge \neg\, wait' \wedge v' = v) \sqsubseteq$
$\exists\, v0\,. \left(\begin{array}{l} R\,(true \vdash true \wedge tr0 = tr \wedge \neg\, wait0) \\ \wedge\, (R(true \vdash tr' = tr0 \wedge \neg\, wait' \wedge v' = v0)) \end{array} \right)$
= [Designs And True B.2.46 and Predicate Calculus]
$R(true \vdash tr' = tr \wedge \neg\, wait' \wedge v' = v) \sqsubseteq$
$\exists\, v0\,. \left(R\,(true \vdash tr' = tr \wedge \neg\, wait0 \wedge \neg\, wait' \wedge v' = v0) \right)$
= [Predicate Calculus]
$R(true \vdash tr' = tr \wedge \neg\, wait' \wedge v' = v) \sqsubseteq$
$\exists\, v\,. \left(R\,(true \vdash tr' = tr \wedge \neg\, wait0 \wedge \neg\, wait' \wedge v' = v) \right)$
= [Factor $\neg$ wait0 makes the right side stronger than the left side, thus it is a refinement]
= true
□

B.3.4 Sequence

Attached Rule 7.

Sequence Progress:

$$\frac{(c_1 \mid s_1 \models A_1) \xrightarrow{l} (c_2 \mid s_2 \models A_2)}{(c_1 \mid s_1 \models A_1 \; ; B) \xrightarrow{l} (c_2 \mid s_2 \models A_2 \; ; B)}$$

Proof

We will divide the proof in two cases, both of which have c_1 and c_2 being true. In the case that some of them is false, the antecedent of the implication is false, what makes the expression directly true.

Case 1 : l = τ and c_1 and c_2 are true

The assumption

$(c_1 \mid s_1 \models A_1) \xrightarrow{l} (c_2 \mid s_2 \models A_2)$

equals (by the definition of Silent Transition B.25)

$\forall$ w . $c_1 \wedge c_2 \Rightarrow$ Lift (s_1) ; $A_1 \sqsubseteq$ Lift (s_2) ; A_2

Abstracting w, as it is universally quantified, we have

$c_1 \wedge c_2 \Rightarrow$ Lift (s_1) ; $A_1 \sqsubseteq$ Lift (s_2) ; A_2

Having c_1 and c_2 being true, by predicate calculus, we have:

[Derived Assumption]: Lift (s_1) ; $A_1 \sqsubseteq$ Lift (s_2) ; A_2

Now we start proving:

$(c_1 \mid s_1 \models A_1 \; ; B) \xrightarrow{\tau} (c_2 \mid s_2 \models A_2 \; ; B)$
= [Definition B.25, of τ transition]
$\forall$ w . $c_1 \wedge c_2 \Rightarrow$ Lift (s_1) ; A_1 ; B $\sqsubseteq$ Lift (s_2) ; A_2 ; B

= [w is universally quantified, so we abstract it]
$c_1 \wedge c_2 \Rightarrow \text{Lift } (s_1)\ ;\ A_1\ ;\ B \sqsubseteq \text{Lift } (s_2)\ ;\ A_2\ ;\ B$
= [Derived Assumption with Monotonicity of Refinement (6) lead us to: Lift (s_1) ; A_1 ; B $\sqsubseteq$ Lift (s_2) ; A_2 ; B]
= true

Case 2 : l is not silent ($\neq \tau$) and c1 and c2 are true
The assumption

$$(c_1 \mid s_1 \models A_1) \xrightarrow{l} (c_2 \mid s_2 \models A_2)$$

equals (by the definition of Labelled Transition B.26)

$\forall\ w\ .\ c_1 \wedge c_2 \Rightarrow \text{Lift } (s_1)\ ;\ A_1 \sqsubseteq (\text{Lift } (s_2)\ ;\ c.w1 \rightarrow A_2) \Box (\text{Lift } (s_1)\ ;\ A_1)$
= [w is universally quantified, so we abstract it]
$c_1 \wedge c_2 \Rightarrow \text{Lift } (s_1)\ ;\ A_1 \sqsubseteq (\text{Lift } (s_2)\ ;\ c.w1 \rightarrow A_2) \Box (\text{Lift } (s_1)\ ;\ A_1)$

Having c_1 and c_2 being true, by predicate calculus, we have:

$\forall\ w\ .\ \text{Lift } (s_1)\ ;\ A_1 \sqsubseteq (\text{Lift } (s_2)\ ;\ c.w1 \rightarrow A_2) \Box (\text{Lift } (s_1)\ ;\ A_1)$

and this lead us to

[Derived Assumption 2]:
$\text{Lift } (s_1)\ ;\ A_1 \sqsubseteq (\text{Lift } (s_2)\ ;\ c.w1 \rightarrow A_2) \Box (\text{Lift } (s_1)\ ;\ A_1)$

Expression to prove:
$(c_1 \mid s_1 \models A_1\ ;\ B) \xrightarrow{l} (c_2 \mid s_2 \models A_2\ ;\ B) =$
$\forall\ w\ .\ c_1 \wedge c_2 \Rightarrow (\text{Lift } (s_1)\ ;\ A_1\ ;\ B) \sqsubseteq (\text{Lift } (s_2)\ ;\ c.w1 \rightarrow A_2\ ;\ B) \Box (\text{Lift } (s_1)\ ;\ A_1\ ;\ B)$
= [w is universally quantified, so we abstract it]
$c_1 \wedge c_2 \Rightarrow (\text{Lift } (s_1)\ ;\ A_1\ ;\ B) \sqsubseteq (\text{Lift } (s_2)\ ;\ c.w1 \rightarrow A_2\ ;\ B) \Box (\text{Lift } (s_1)\ ;\ A_1\ ;\ B)$
= [External Choice/Sequence Distributive B.2.11]
$c_1 \wedge c_2 \Rightarrow (\text{Lift } (s_1)\ ;\ A_1\ ;\ B) \sqsubseteq ((\text{Lift } (s_2)\ ;\ c.w1 \rightarrow A_2) \Box (\text{Lift } (s_1)\ ;\ A_1)\ ;\ B)$
= [Derived Assumption 2 with Monotonicity of Refinement (6) lead us to: (Lift (s_1) ; A_1 ; B) $\sqsubseteq$ (Lift (s_2) ; c.w1 $\rightarrow$ A_2 ; B) $\Box$ (Lift (s_1) ; A_1 ; B)]
$c_1 \wedge c_2 \Rightarrow \text{true}$

= [(P $\Rightarrow$ true) = true]
= true

□

Attached Rule 8.

***Sequence End**:*

$$\frac{}{(c \mid s \models \mathit{Skip}\ ;\ A) \xrightarrow{\tau} (c \mid s \models A)}\ c$$

Proof $(c \mid s \models \mathit{Skip}\ ;\ A) \xrightarrow{\tau} (c \mid s \models A)$
= [Theorem B.2.2]
$(c \mid s \models A) \xrightarrow{\tau} (c \mid s \models A)$
= [Theorem 2]
true □

B.3.5 Internal Choice*

Attached Rule 9.

Internal Choice Left*:*
$(c \mid s \models A_1 \sqcap A_2) \xrightarrow{\tau} (c \mid s \models A_1)$

Proof
$(c \mid s \models A_1 \sqcap A_2) \xrightarrow{\tau} (c \mid s \models A_1)$
= [Definition of Silent Transition B.25]
$\forall\, w \,.\, c \wedge c \Rightarrow \text{Lift}\ (s)\ ;\ (A_1 \sqcap A_2) \sqsubseteq \text{Lift}\ (s)\ ;\ A_1$
= [w is universally quantified, so we abstract it]
$c \wedge c \Rightarrow \text{Lift}\ (s)\ ;\ (A_1 \sqcap A_2) \sqsubseteq \text{Lift}\ (s)\ ;\ A_1$
= [Refinement of Non-Determinism B.2.25]
$c \wedge c \Rightarrow \text{Lift}\ (s)\ ;\ A_1 \sqsubseteq \text{Lift}\ (s)\ ;\ A_1$
= [Refinement Equal sides 1]
$c \wedge c \Rightarrow \text{true}$
= [Predicate Calculus]
= true □

Attached Rule 10.

Internal Choice Right*:*
$(c \mid s \models A_1 \sqcap A_2) \xrightarrow{\tau} (c \mid s \models A_2)$

Proof
$(c \mid s \models A_1 \sqcap A_2) \xrightarrow{\tau} (c \mid s \models A_2)$
= Definition of Silent Transition B.25
$\forall\, w \,.\, c \wedge c \Rightarrow \text{Lift}\ (s)\ ;\ (A_1 \sqcap A_2) \sqsubseteq \text{Lift}\ (s)\ ;\ A_2$
= [w is universally quantified, so we abstract it]
$c \wedge c \Rightarrow \text{Lift}\ (s)\ ;\ (A_1 \sqcap A_2) \sqsubseteq \text{Lift}\ (s)\ ;\ A_2$
= [Refinement Non-Determinism B.2.25]
$c \wedge c \Rightarrow \text{Lift}\ (s)\ ;\ A_1 \sqsubseteq \text{Lift}\ (s)\ ;\ A_2$
= [Refinement Equal sides 1]
$c \wedge c \Rightarrow \text{true}$
= [Predicate Calculus]
= true □

B.3.6 Guard

Attached Rule 11.

***Guard**:*

$$\frac{c \qquad (s\ ;\ g)}{(c \mid s \models g\ \&\ A) \xrightarrow{\tau} (c \wedge (s\ ;\ g) \mid s \models A)}$$

Proof

[Case 1: g is true]

$(c \mid s \models g\ \&\ A) \xrightarrow{\tau} (c \wedge (s\ ;\ g) \mid s \models A)$

= [Definition of Silent Transition B.25]

$\forall\ w\ .\ c \wedge c \wedge s\ ;\ g \Rightarrow (\text{Lift}\ (s)\ ;\ g\ \&\ A) \sqsubseteq (\text{Lift}\ (s)\ ;\ A)$

= [w is universally quantified, so we abstract it]

$c \wedge c \wedge s\ ;\ g \Rightarrow (\text{Lift}\ (s)\ ;\ g\ \&\ A) \sqsubseteq (\text{Lift}\ (s)\ ;\ A)$

= [Assumptions c and s ; g]

$(\text{true} \wedge \text{true} \wedge \text{true}) \Rightarrow (\text{Lift}\ (s)\ ;\ g\ \&\ A) \sqsubseteq (\text{Lift}\ (s)\ ;\ A)$

= [Predicate Calculus]

$(\text{Lift}\ (s)\ ;\ g\ \&\ A) \sqsubseteq (\text{Lift}\ (s)\ ;\ A)$

[True guard B.2.4]

$(\text{Lift}\ (s)\ ;\ A) \sqsubseteq (\text{Lift}\ (s)\ ;\ A)$

= [Refinement Equal Sides 1]

true

[Case 2: g is false]

$(c \mid s \models g\ \&\ A) \xrightarrow{\tau} (c \wedge (s\ ;\ g) \mid s \models A)$

= [Definition of Silent Transition B.25]

$\forall\ w\ .\ c \wedge c \wedge s\ ;\ g \Rightarrow (\text{Lift}\ (s)\ ;\ g\ \&\ A) \sqsubseteq (\text{Lift}\ (s)\ ;\ A)$

= [w is universally quantified, so we abstract it]

$c \wedge c \wedge s\ ;\ g \Rightarrow (\text{Lift}\ (s)\ ;\ g\ \&\ A) \sqsubseteq (\text{Lift}\ (s)\ ;\ A)$

= [Case 2: g is false (replacing g by false)]

$c \wedge c \wedge s\ ;\ \text{false} \Rightarrow (\text{Lift}\ (s)\ ;\ \text{false}\ \&\ A) \sqsubseteq (\text{Lift}\ (s)\ ;\ A)$

= [S Sequence False Equals False 5]

$c \wedge c \wedge \text{false} \Rightarrow (\text{Lift}\ (s)\ ;\ \text{false}\ \&\ A) \sqsubseteq (\text{Lift}\ (s)\ ;\ A)$

= [Predicate Calculus]

$\text{false} \Rightarrow (\text{Lift}\ (s)\ ;\ \text{false}\ \&\ A) \sqsubseteq (\text{Lift}\ (s)\ ;\ A)$

= [Predicate Calculus]

true

□

B.3.7 External Choice*

Attached Rule 12.

External Choice Begin*:*
$(c \mid s \models A_1 \,\Box\, A_2) \xrightarrow{\tau} (c \mid s \models (loc\ s \bullet A_1) \boxplus (loc\ s \bullet A_2))$

Proof
$\forall$ w . c $\wedge$ c $\Rightarrow$ (Lift (s) ; $A_1 \,\Box\, A_2 \sqsubseteq$ (loc s $\bullet$ A_1) $\boxplus$ (loc s $\bullet$ A_2))
= [Definition of Extra Choice B.30]
$\forall$ w . c $\wedge$ c $\Rightarrow$ (Lift (s) ; $A_1 \,\Box\, A_2 \sqsubseteq$ (loc s $\bullet$ A_1) $\Box$ (loc s $\bullet$ A_2))
= [w is universally quantified, so we abstract it]
c $\wedge$ c $\Rightarrow$ (Lift (s) ; $A_1 \,\Box\, A_2 \sqsubseteq$ (loc s $\bullet$ A_1) $\Box$ (loc s $\bullet$ A_2))
= [Definition of loc B.29]
c $\wedge$ c $\Rightarrow$ (Lift (s) ; $A_1 \,\Box\, A_2 \sqsubseteq$ (Lift (s) ; A_1) $\Box$ (Lift (s) ; A_2))
= [Lift External Choice B.2.28]
c $\wedge$ c $\Rightarrow$ (Lift (s) ; $A_1 \,\Box\, A_2 \sqsubseteq$ (Lift (s) ; Lift (s) ; $A_1 \,\Box\, A_2$))
= [Lift Left Unit B.2.29]
c $\wedge$ c $\Rightarrow$ (Lift (s) ; $A_1 \,\Box\, A_2 \sqsubseteq$ (Lift (s) ; ($A_1 \,\Box\, A_2$)))
= [Parenthesis]
c $\wedge$ c $\Rightarrow$ (Lift (s) ; ($A_1 \,\Box\, A_2$) $\sqsubseteq$ (Lift (s) ; ($A_1 \,\Box\, A_2$)))
= [Refinement Equal Sides 1]
c $\wedge$ c $\Rightarrow$ true
= [(P $\Rightarrow$ true) = true]
true
$\Box$

Attached Rule 13.

External Choice Skip*:*
$(c \mid s \models loc\ s_1 \bullet Skip \boxplus loc\ s_2 \bullet A) \xrightarrow{\tau} (c \mid s \models Skip)$

Proof (c | s $\models$ loc s_1 $\bullet$ Skip $\boxplus$ loc s_2 $\bullet$ A) $\xrightarrow{\tau}$ (c | s_1 $\models$ Skip)
= [Definition of Silent Transition B.25]
$\forall$ w . c $\Rightarrow$ Lift (s_1) ; (loc s_1 $\bullet$ Skip $\boxplus$ loc s_2 $\bullet$ A) $\sqsubseteq$ Lift (s_1) ; Skip
= [w is universally quantified, so we abstract it]
c $\Rightarrow$ Lift (s_1) ; (loc s_1 $\bullet$ Skip $\boxplus$ loc s_2 $\bullet$ A) $\sqsubseteq$ Lift (s_1) ; Skip
= [Definition of Loc B.29]

$c \Rightarrow \text{Lift}\ (s_1)\ ;\ (\text{Lift}\ (s_1)\ ;\ \text{Skip} \boxplus \text{Lift}\ (s_2)\ ;\ A) \sqsubseteq \text{Lift}\ (s_1)\ ;\ \text{Skip}$

= [Lift is CSP4 B.2.26]

$c \Rightarrow \text{Lift}\ (s_1)\ ;\ (\text{Lift}\ (s_1) \boxplus \text{Lift}\ (s_2)\ ;\ A) \sqsubseteq \text{Lift}\ (s_1)\ ;\ \text{Skip}$

= [Extra Choice Definition B.30]

$c \Rightarrow \text{Lift}\ (s_1)\ ;\ (\text{Lift}\ (s_1) \Box \text{Lift}\ (s_2)\ ;\ A) \sqsubseteq \text{Lift}\ (s_1)\ ;\ \text{Skip}$

= [External Choice Assignment]

$c \Rightarrow \text{Lift}\ (s_1)\ ;\ \text{Lift}\ (s_1) \sqsubseteq \text{Lift}\ (s_1)\ ;\ \text{Skip}$

= [Lift is CSP4 B.2.26]

$c \Rightarrow \text{Lift}\ (s_1)\ ;\ \text{Lift}\ (s_1) \sqsubseteq \text{Lift}\ (s_1)$

= [Lift Left Unit B.2.29]

$c \Rightarrow \text{Lift}\ (s_1) \sqsubseteq \text{Lift}\ (s_1)$

= [Refinement Equal Sides 1]

$c \Rightarrow \text{true}$

= [Predicate Calculus]

$= \text{true}$

□

Attached Rule 14.

External Choice End*:

$$\frac{(c_1 \mid s_1 \models A_1) \xrightarrow{l} (c_3 \mid s_3 \models A_3)}{(c \mid s \models \mathit{loc}\ s_1 \bullet A_1 \boxplus \mathit{loc}\ s_2 \bullet A_2) \xrightarrow{l} (c_3 \mid s_3 \models A_3)}$$

Proof

$(c \mid s \models \text{loc}\ s_1 \bullet A_1 \boxplus \text{loc}\ s_2 \bullet A_2) \xrightarrow{l} (c_3 \mid s_3 \models A_3)$

= [Definition of labelled transition B.26]

$\forall\ w\ .\ c \wedge c_3 \Rightarrow \text{Lift}\ (s)\ ;\ ((\text{loc}\ s_1 \bullet A_1) \boxplus (\text{loc}\ s_2 \bullet A_2)) \sqsubseteq (\text{Lift}\ (s_3)\ ;\ l \rightarrow A_3) \Box (\text{Lift}\ (s)\ ;\ ((\ \text{loc}\ s_1 \bullet A_1\ ) \boxplus (\ \text{loc}\ s_2 \bullet A_2)))$

= [w is universally quantified, so we abstract it]

$c \wedge c_3 \Rightarrow \text{Lift}\ (s)\ ;\ ((\text{loc}\ s_1 \bullet A_1) \boxplus (\text{loc}\ s_2 \bullet A_2)) \sqsubseteq (\text{Lift}\ (s_3)\ ;\ l \rightarrow A_3) \Box (\text{Lift}\ (s)\ ;\ ((\ \text{loc}\ s_1 \bullet A_1\ ) \boxplus (\ \text{loc}\ s_2 \bullet A_2)))$

= [Definition of Extra Choice B.30]

$c \wedge c_3 \Rightarrow \text{Lift}\ (s)\ ;\ ((\text{loc}\ s_1 \bullet A_1) \Box (\text{loc}\ s_2 \bullet A_2)) \sqsubseteq (\text{Lift}\ (s_3)\ ;\ l \rightarrow A_3) \Box (\text{Lift}\ (s)\ ;\ ((\ \text{loc}\ s_1 \bullet A_1\ ) \Box (\ \text{loc}\ s_2 \bullet A_2)))$

= [Definition of Loc B.29]

$c \wedge c_3 \Rightarrow \text{Lift}\ (s)\ ;\ ((\text{loc}\ s_1 \bullet A_1) \Box (\text{loc}\ s_2 \bullet A_2)) \sqsubseteq (\text{Lift}\ (s_3)\ ;\ l \rightarrow A_3) \Box (\text{Lift}\ (s)\ ;\ ((\text{Lift}\ (s_1)\ ;\ A_1) \Box (\text{Lift}\ (s_2)\ ;\ A_2)))$

= [Lift External Choice B.2.28]

$c \wedge c_3 \Rightarrow$ Lift (s) ; ((loc $s_1 \bullet A_1$) $\Box$ (loc $s_2 \bullet A_2$)) $\sqsubseteq$ (Lift (s_3) ; $l \rightarrow A_3$) $\Box$ (Lift (s) ; Lift (s_1) ; A_1) $\Box$ (Lift (s) ; Lift (s_2) ; A_2)))

= [Lift Left Unit B.2.29]

$c \wedge c_3 \Rightarrow$ Lift (s) ; ((loc $s_1 \bullet A_1$) $\Box$ (loc $s_2 \bullet A_2$)) $\sqsubseteq$ (Lift (s_3) ; $l \rightarrow A_3$) $\Box$ (Lift (s_1) ; A_1) $\Box$ (Lift (s_2) ; A_2)))

= [Assumption]

$c \wedge c_3 \Rightarrow$ Lift (s) ; ((loc $s_1 \bullet A_1$) $\Box$ (loc $s_2 \bullet A_2$)) $\sqsubseteq$ (Lift (s_1) ; A_1) $\Box$ (Lift (s_2) ; A_2)))

= [Lift Left Unit B.2.29]

$c \wedge c_3 \Rightarrow$ Lift (s) ; ((loc $s_1 \bullet A_1$) $\Box$ (loc $s_2 \bullet A_2$)) $\sqsubseteq$ (Lift (s) ; Lift (s_1) ; A_1) $\Box$ (Lift (s) ; Lift (s_2) ; A_2)))

= [Lift External Choice B.2.28]

$c \wedge c_3 \Rightarrow$ Lift (s) ; ((loc $s_1 \bullet A_1$) $\Box$ (loc $s_2 \bullet A_2$)) $\sqsubseteq$ (Lift (s) ; (Lift (s_1) ; A_1) $\Box$ Lift (s_2) ; A_2)

= [Definition of Loc B.29]

$c \wedge c_3 \Rightarrow$ Lift (s) ; ((loc $s_1 \bullet A_1$) $\Box$ (loc $s_2 \bullet A_2$)) $\sqsubseteq$ (Lift (s) ; (loc $s_1 \bullet A_1$) $\Box$ loc $s_2 \bullet A_2$)

= [Definition of Extra Choice B.30]

$c \wedge c_3 \Rightarrow$ Lift (s) ; ((loc $s_1 \bullet A_1$) $\Box$ (loc $s_2 \bullet A_2$)) $\sqsubseteq$ (Lift (s) ; (loc $s_1 \bullet A_1$) $\boxplus$ loc $s_2 \bullet A_2$)

= [Refinement Equal Sides 1]

$c \wedge c_3 \Rightarrow$ true

= [Predicate Calculus]

= true

$\Box$

Attached Rule 15.

External Choice Silent Left:

$$\frac{(c_1 \mid s_1 \models A_1) \xrightarrow{\tau} (c_3 \mid s_3 \models A_3)}{(c \mid s \models loc\ s_1 \bullet A_1 \boxplus loc\ s_2 \bullet A_2) \xrightarrow{\tau} (c \mid s \models loc\ s_3 \bullet A_3 \boxplus loc\ s_2 \bullet A_2)}$$

(c1 | s $\models$ (loc s1 $\bullet$ A1) $\boxplus$ (loc s2 $\bullet$ A2))

$\xrightarrow{\tau}$

(c2 | s $\models$ (loc s3 $\bullet$ A3) $\boxplus$ (loc s2 $\bullet$ A2))

= [loc definition B.29]

(c1 | s $\models$ (Lift (s1); A1) $\boxplus$ (Lift (s2); A2))

$\xrightarrow{\tau}$

(c2 | s $\models$ (Lift (s3); A3) $\boxplus$ (Lift (s2); A2))

= [Extra choice definition B.30]

(c1 ∣ s ⊨ (Lift (s1); A1) □ (Lift (s2); A2))

$\xrightarrow{\tau}$

(c2 ∣ s ⊨ (Lift (s3); A3) □ (Lift (s2); A2))

= [Definition of Silent Transition B.25]

∀w . c1 ∧ c2 ⇒ (Lift (s1); A1) □ (Lift (s2); A2)) ⊑ (Lift (s3); A3) □ (Lift (s2); A2)

= [w is universally quantified, so we abstract it]

c1 ∧ c2 ⇒ (Lift (s1); A1) □ (Lift (s2); A2)) ⊑ (Lift (s3); A3) □ (Lift (s2); A2)

[Assumpt. (Lift (s1) ; A1 ⊑ Lift (s3) ; A3) + Ref.Eq.Sides (1)]

c1 ∧ c2 ⇒ true

= [Predicate Calculus]

true

Attached Rule 16.

***External Choice Silent Right**:*

$$\frac{(c_2 \mid s_2 \models A_2) \xrightarrow{\tau} (c_3 \mid s_3 \models A_3)}{(c \mid s \models loc\ s_1 \bullet A_1 \boxplus loc\ s_2 \bullet A_2) \xrightarrow{\tau} (c \mid s \models loc\ s_1 \bullet A_1 \boxplus loc\ s_3 \bullet A_3)}$$

(c1 ∣ s ⊨ (loc s1 • A1) ⊞ (loc s2 • A2))

$\xrightarrow{\tau}$

(c2 ∣ s ⊨ (loc s1 • A1) ⊞ (loc s3 • A3))

= [loc definition B.29]

(c1 ∣ s ⊨ (Lift (s1); A1) ⊞ (Lift (s2); A2))

$\xrightarrow{\tau}$

(c2 ∣ s ⊨ (Lift (s1); A1) ⊞ (Lift (s3); A3))

= [Extra choice definition B.30]

(c1 ∣ s ⊨ (Lift (s1); A1) □ (Lift (s2); A2))

$\xrightarrow{\tau}$

(c2 ∣ s ⊨ (Lift (s1); A1) □ (Lift (s3); A3))

= [Definition of Silent Transition B.25]

∀w . c2 ∧ c2 ⇒ (Lift (s1); A1) □ (Lift (s2); A2)) ⊑ (Lift (s1); A1) □ (Lift (s3); A3)

= [w is universally quantified, so we abstract it]

c2 ∧ c2 ⇒ (Lift (s1); A1) □ (Lift (s2); A2)) ⊑ (Lift (s1); A1) □ (Lift (s3); A3)

= [Lift (s2) ; A2 ⊑ Lift (s3) ; A3 (this is the assumption) + Refinement Equal Sides 1]

c1 ∧ c2 ⇒ true

= [Predicate Calculus]
true

B.3.8 Parallelism*

Attached Rule 17.

Parallel Begin*:*

$(c \mid s \models A_1 \;[\![\, s_1 \mid cs \mid s_2 \,]\!]\; A_2)$
$\xrightarrow{\tau}$
$(c \mid s \models (loc\ (s \mid x_1)_{+x2} \bullet A_1) \;[\![\, x_1 \mid cs \mid x_2 \,]\!]\; (loc\ (s \mid x_2)_{+x1} \bullet A_2))$

Proof

$(c \mid s \models A_1 \;[\![\, s_1 \mid cs \mid s_2 \,]\!]\; A_2)$
$\xrightarrow{\tau}$
$(c \mid s \models (loc\ (s \mid x_1)_{+x2} \bullet A_1) \;[\![\, x_1 \mid cs \mid x_2 \,]\!]\; (loc\ (s \mid x_2)_{+x1} \bullet A_2))$
= [Definition of Silent Transition B.25]
$\forall\, w\ .\ c \Rightarrow Lift\ (s)\ ;\ A_1\ |[\ s_1 \mid cs \mid s_2\ ]|\ A_2 \sqsubseteq Lift\ (s)\ ;\ (loc\ (s \mid x_1)_{+x2} \bullet A_1) \;[\![\, x_1 \mid cs \mid x_2 \,]\!]$
$(loc\ (s \mid x_2)_{+x1} \bullet A_2$
The proof from now on will consist on proving that the RHS of the refinement is equal to the LHS.
RHS =
$Lift\ (s)\ ;\ (loc\ (s \mid x_1)_{+x2} \bullet A_1) \;[\![\, x_1 \mid cs \mid x_2 \,]\!]\; (loc\ (s \mid x_2)_{+x1} \bullet A_2$
= [Definition of Loc B.29]
$Lift\ (s)\ ;\ (Lift\ (s \mid x_1)_{+x2}\ ;\ A_1) \;[\![\, x_1 \mid cs \mid x_2 \,]\!]\; (Lift\ (s \mid x_2)_{+x1}\ ;\ A_2$
= [Parallel Distributivity B.2.37]
$Lift\ (s)\ ;\ (Lift\ (s \mid x_1)_{+x2}\ ;\ A_1) \;[\![\, x_1 \mid cs \mid x_2 \,]\!]\; Lift\ (s)\ ;\ (Lift\ (s \mid x_2)_{+x1}\ ;\ A_2$
= [Lift Composition (twice)B.2.27]
$Lift\ (s;\ (s \mid x_1)_{+x2}\ ;\ A_1) \;[\![\, x_1 \mid cs \mid x_2 \,]\!]\; Lift\ (s\ ;\ (s \mid x_2)_{+x1}\ ;\ A_2$
= [Lift Semi Idempotence B.2.38]
$Lift\ (s)\ ;\ A_1) \;[\![\, x_1 \mid cs \mid x_2 \,]\!]\; Lift\ (s_2)\ ;\ A_2$
= [Lift Parallel Distributivity B.2.37]
$Lift\ (s)\ ;\ A_1 \;[\![\, x_1 \mid cs \mid x_2 \,]\!]\; A_2$
= LHS □

Attached Rule 18.

Parallel End*:*

$(c \mid s \models (loc\ s_1 \bullet Skip) \;[\![\, x_1 \mid cs \mid x_2 \,]\!]\; (loc\ s_2 \bullet Skip)) \xrightarrow{\tau} (c \mid (s_1 \mid x1) \wedge (s_2 \mid x_2) \models Skip)$

Proof

$(c \mid s \models (loc\ s_1 \bullet Skip) \;[\![\, x_1 \mid cs \mid x_2 \,]\!]\; (loc\ s_2 \bullet Skip)) \xrightarrow{\tau} (c \mid (s_1 \mid x_1) \wedge (s_2 \mid x_2) \models Skip)$

= [Definition of Silent Transiton B.25]

$$\forall \text{ w . c} \Rightarrow \left(\begin{array}{l} \mathit{Lift}\,(s)\,;\ \mathit{loc}\,s_1 \bullet \mathit{Skip} \ [\![\mathit{cs}]\!]\ \mathit{loc}\,s_2 \bullet \mathit{Skip} \\ \sqsubseteq \mathit{Lift}\,(s)\,;\ (\mathit{loc}\,(s \mid x_1)_{+x2} \bullet \mathit{Skip}) \ [\![x_1 \mid \mathit{cs} \mid x_2]\!]\ (\mathit{loc}\,(s \mid x_2)_{+x1} \bullet \mathit{Skip}) \end{array} \right)$$

The proof from now on will consist on proving that the RHS of the refinement is equal to the LHS.

LHS

= Lift (s) ; (loc s_1 • Skip) [[x_1 | cs | x_2]] (loc s_2 • Skip)

= [Definition of Loc B.29]

Lift (s) ; (Lift (s_1) ; Skip) [[x_1 | cs | x_2]] (Lift (s_2) ; Skip)

= [Parallel Distributivity B.2.37]

(Lift (s) ; Lift(s_1) ; Skip) [[x_1 | cs | x_2]] (Lift (s) ; Lift(s_2) ; Skip)

= [Lift Left Unit B.2.29 twice]

(Lift (s_1) ; Skip) [[x_1 | cs | x_2]] (Lift (s_2) ; Skip)

= [Lift CSP4 B.2.26 twice]

Lift (s_1) [[x_1 | cs | x_2]] Lift (s_2)

= [Lift Merge B.2.39]

Lift (s_1 | x_1 ∧ s_2 | x_2) ; Skip

=

RHS □

Attached Rule 19.

Parallel Independent Left:

$$\frac{(c_1 \mid s_1 \models A_1) \xrightarrow{l} (c_3 \mid s_3 \models A_3) \ \ ((l = \tau) \vee \mathit{chan}(l) \notin \mathit{cs})}{\begin{array}{l}(c \mid s \models \mathit{loc}\ s_1 \bullet A_1 \ [\![x_1 \mid \mathit{CS} \mid x_2]\!]\ \mathit{loc}\ s_2 \bullet A_2) \\ \xrightarrow{l} (c \mid s \models \mathit{loc}\ s_3 \bullet A_3 \ [\![x_1 \mid \mathit{CS} \mid x_2]\!]\ \mathit{loc}\ s_2 \bullet A_2)\end{array}}$$

Recapitulating the assumptions given on rule 19, they are:

- (c_1 | s_1 ⊨ A_1) $\xrightarrow{l}$ (c_3 | s_3 ⊨ A_3)
- ((l = τ) ∨ chan(l) ∉ cs)

Beyond the previous two assumptions, we also provide the following assumptions (in order to apply Parallel Prefixed Loc 8 rule):

- Lift (s1) ; A1 ⊑ Lift (s3) ; l → A3 □ Lift (s1) ; A1 **[Assumption 3]**
- initials (Lift (s2) ; A2) ⊆ CS **[Assumption 4]**
- CS ∩ usedC (Lift (s3); A3) = ∅ **[Assumption 5]**
- wrtV (Lift (s3); A3) ∩ usedV (Lift (s2) ; A2) = ∅ **[Assumption 6]**

- Lift (s2) ; A2 is divergence free **[Assumption 7]**
- ok = ok' **[Assumption 8]**

From assumptions (c ⌈ s1 $\models$ A1) $\xrightarrow{l}$ (c3 ⌈ s3 $\models$ A3) and (l $\neq \tau$)

we apply Labelled Transition Definition and find

$\forall$ w . c $\wedge$ c3 $\Rightarrow$ Lift (s1) ; A1 $\sqsubseteq$ Lift (s3) ; A3

Then we consider both c and c3 being true and we find

$\forall$ w . true $\wedge$ true $\Rightarrow$ Lift (s1) ; A1 $\sqsubseteq$ Lift (s3) ; A3

then we apply Predicate Calculus twice and find

Lift (s1) ; A1 $\sqsubseteq$ Lift (s3) ; A3

For the case where c or c3 is false, we have a false assumption and then it makes the expression directly true ((false $\Rightarrow$ Rule) = true).

Proof
As there is an $\vee$ operator between factor l = τ and factor chan (l) $\notin$ CS, the proof must be done to both cases:

For l = τ:

(c ⌈ s $\models$ loc s_1 • A_1 $[\![x_1 \mid CS \mid x_2]\!]$ loc s_2 • A_2)
$\xrightarrow{l}$ (c3 ⌈ s $\models$ loc s_3 • A_3 $[\![x_1 \mid CS \mid x_2]\!]$ loc s_2 • A_2)
= [Definition of Silent Transition B.25 (as l = τ, the transition is silent)]
$\forall$ w . c $\wedge$ c3 $\Rightarrow$
Lift (s) ; (loc s1 • A1) $[\![$ x1 {| CS |} x2 $]\!]$ (loc s2 • A2)
$\sqsubseteq$ Lift (s) ; (loc s3 • A3) $[\![$ x1 ⌈ CS ⌈ x2 $]\!]$ (loc s2 • A2)
= [w is universally quantified, so we abstract it throughout the proof]
c $\wedge$ c3 $\Rightarrow$
Lift (s) ; (loc s1 • A1) $[\![$ x1 {| CS |} x2 $]\!]$ (loc s2 • A2)
$\sqsubseteq$ Lift (s) ; (loc s3 • A3) $[\![$ x1 ⌈ CS ⌈ x2 $]\!]$ (loc s2 • A2)

= [Par.Ind.Ref.9 (providing Assumpt.1 from the rule)+Ref.Eq.Sides 1+Ref.Mon.6]
true

For chan (l) $\notin$ CS , l $\neq \tau$ and c and c3 are true
(when at least some of them is false the whole expression is true):

$(c \mid s \models \text{loc } s_1 \bullet A_1 \; [\![x_1 \mid CS \mid x_2]\!] \; \text{loc } s_2 \bullet A_2)$
$\xrightarrow{l} (c3 \mid s \models \text{loc } s_3 \bullet A_3 \; [\![x_1 \mid CS \mid x_2]\!] \; \text{loc } s_2 \bullet A_2)$
= [Definition of Labelled Transition B.26 (as l $\neq \tau$, the transition is labelled)]
$\forall$ w . c $\wedge$ c3 $\Rightarrow$
$$\left(\begin{array}{l} Lift(s);\ locs_1 \bullet A_1 \; [\![x_1 \mid CS \mid x_2]\!] \; locs_2 \bullet A_2 \\ \sqsubseteq \\ \left(\begin{array}{l} Lift(s);\ l \rightarrow locs_3 \bullet A_3 \; [\![x_1 \mid CS \mid x_2]\!] \; locs_2 \bullet A_2 \\ \Box\ Lift(s);\ locs_1 \bullet A_1 \; [\![x_1 \mid CS \mid x_2]\!] \; locs_2 \bullet A_2 \end{array} \right) \end{array} \right)$$
= [w is universally quantified, so we abstract it]
c $\wedge$ c3 $\Rightarrow$
$$\left(\begin{array}{l} Lift(s);\ locs_1 \bullet A_1 \; [\![x_1 \mid CS \mid x_2]\!] \; locs_2 \bullet A_2 \\ \sqsubseteq \\ \left(\begin{array}{l} Lift(s);\ l \rightarrow locs_3 \bullet A_3 \; [\![x_1 \mid CS \mid x_2]\!] \; locs_2 \bullet A_2 \\ \Box\ Lift(s);\ locs_1 \bullet A_1 \; [\![x_1 \mid CS \mid x_2]\!] \; locs_2 \bullet A_2 \end{array} \right) \end{array} \right)$$
= [c and c3]
true $\Rightarrow$
$$\left(\begin{array}{l} Lift(s);\ locs_1 \bullet A_1 \; [\![x_1 \mid CS \mid x_2]\!] \; locs_2 \bullet A_2 \\ \sqsubseteq \\ \left(\begin{array}{l} Lift(s);\ l \rightarrow locs_3 \bullet A_3 \; [\![x_1 \mid CS \mid x_2]\!] \; locs_2 \bullet A_2 \\ \Box\ Lift(s);\ locs_1 \bullet A_1 \; [\![x_1 \mid CS \mid x_2]\!] \; locs_2 \bullet A_2 \end{array} \right) \end{array} \right)$$
= [Predicate Calculus]
$$\left(\begin{array}{l} Lift(s);\ locs_1 \bullet A_1 \; [\![x_1 \mid CS \mid x_2]\!] \; locs_2 \bullet A_2 \\ \sqsubseteq \\ \left(\begin{array}{l} Lift(s);\ l \rightarrow locs_3 \bullet A_3 \; [\![x_1 \mid CS \mid x_2]\!] \; locs_2 \bullet A_2 \\ \Box\ Lift(s);\ locs_1 \bullet A_1 \; [\![x_1 \mid CS \mid x_2]\!] \; locs_2 \bullet A_2 \end{array} \right) \end{array} \right)$$
= [External Choice Refinement Implication 15]
$$\left(\begin{array}{l} Lift(s);\ locs_1 \bullet A_1 \; [\![x_1 \mid CS \mid x_2]\!] \; locs_2 \bullet A_2 \\ \sqsubseteq \\ \left(\begin{array}{l} Lift(s);\ l \rightarrow locs_3 \bullet A_3 \; [\![x_1 \mid CS \mid x_2]\!] \; locs_2 \bullet A_2 \end{array} \right) \end{array} \right)$$
= [Parallel Prefixed Loc 8 (having assumptions 3 to 8) with Monotonicity of Refinement 6]

true
□

Attached Rule 20.

Parallel Independent Right:

$$\frac{(c_2 \mid s_2 \models A_2) \xrightarrow{l} (c_3 \mid s_3 \models A_3)\ ((l = \tau) \vee chan(l) \notin cs)}{(c \mid s \models loc\ s_1 \bullet A_1 \ [\![x_1 \mid CS \mid x_2]\!]\ loc\ s_2 \bullet A_2) \xrightarrow{l} (c \mid s \models loc\ s_1 \bullet A_1 \ [\![x_1 \mid CS \mid x_2]\!]\ loc\ s_3 \bullet A_3)}$$

Recapitulating the previous assumptions:

- $(c_2 \mid s_2 \models A_2) \xrightarrow{l} (c_3 \mid s_3 \models A_3)$ **[Assumption 1]**
- $(l = \tau) \vee chan(l) \notin cs$ **[Assumption 2]**

Beyond the previous two assumptions, we also provide the following assumptions (in order to apply Parallel Prefixed Loc 8 rule):

- Lift (s2) ; A2 $\sqsubseteq$ Lift (s3) ; l $\rightarrow$ A3 $\square$ Lift (s2) ; A2 **[Assumption 3]**
- initials (Lift (s1) ; A1) $\subseteq$ CS **[Assumption 4]**
- CS $\cap$ usedC (Lift (s3); A3) = $\emptyset$ **[Assumption 5]**
- wrtV (Lift (s3); A3) $\cap$ usedV (Lift (s1) ; A1) = $\emptyset$ **[Assumption 6]**
- Lift (s1) ; A1 is divergence free **[Assumption 7]**
- ok = ok' **[Assumption 8]**

Proof

$(c \mid s \models loc\ s_1 \bullet A_1 \ [\![x_1 \mid CS \mid x_2]\!]\ loc\ s_2 \bullet A_2)$
$\xrightarrow{l} (c \mid s \models loc\ s_1 \bullet A_1 \ [\![x_1 \mid CS \mid x_2]\!]\ loc\ s_3 \bullet A_3)$
= [Compound Actions Commutative B.2.6]
$(c \mid s \models loc\ s_2 \bullet A_2 \ [\![x_2 \mid CS \mid x_1]\!]\ loc\ s_1 \bullet A_1)$
$\xrightarrow{l} (c \mid s \models loc\ s_1 \bullet A_1 \ [\![x_1 \mid CS \mid x_2]\!]\ loc\ s_3 \bullet A_3)$
= [Compound Actions Commutative B.2.6]
$(c \mid s \models loc\ s_2 \bullet A_2 \ [\![x_2 \mid CS \mid x_1]\!]\ loc\ s_1 \bullet A_1)$
$\xrightarrow{l} (c \mid s \models loc\ s_3 \bullet A_3 \ [\![x_3 \mid CS \mid x_1]\!]\ loc\ s_1 \bullet A_1)$
= [Parallel Independent Left 19 (having assumptions 2-12)]
true □

Attached Rule 21.

Parallel Synchronised:

$$\frac{\begin{array}{l} d \in cs\ c_1\ c_2\ c_3\ c_4\ (w_1 = w_2) \\ (^*,\diamond) \in (?, !), (!, ?), (!, !), (., .), (?, ?) \\ (c_1 \mid s_1 \models A_1) \xrightarrow{d^* w_1} (c_3 \mid s_3 \models A_3) \\ (c_2 \mid s_2 \models A_2) \xrightarrow{d \diamond w_2} (c_4 \mid s_4 \models A_4) \end{array}}{\begin{array}{l} (c_1 \wedge c_2 \mid s \models (c_1 \mid loc\ s_1 \bullet A_1) \ [\![x_1 \mid cs \mid x_2]\!] \ (c_2 \mid loc\ s_2 \bullet A_2)) \\ \xrightarrow{d|w_2} \\ \left(\begin{array}{l} c_3 \wedge c_4 \wedge w_1 = w_2 \mid s \models \\ \left(\begin{array}{l} (c_3 \wedge (w_1 = w_2) \mid locs_3 \bullet A_3) \\ [\![x_1 \mid cs \mid x_2]\!] \\ (c_4 \wedge (w_1 = w_2) \mid locs_4 \bullet A_4) \end{array} \right) \end{array} \right) \end{array}}$$

Recapitulating the previous assumptions:

- $d \in cs, c_1, c_2, c_3, c_4$ **[Assumptions 1, 2, 3, 4 and 5]**
- $w_1 = w_2$ **[Assumption 6]**
- $(^*,\diamond) \in (?, !), (!, ?), (!, !), (., .), (?, ?)$ **[Assumption 7]**
- $(c_1 \mid s_1 \models A_1) \xrightarrow{d^* w_1} (c_3 \mid s_3 \models A_3)$ **[Assumption 8]**
- $(c_2 \mid s_2 \models A_2) \xrightarrow{d \diamond w_2} (c_4 \mid s_4 \models A_4)$ **[Assumption 9]**

We also provide the following assumption in order to allow the proof using Lift Shift 7:

- 3.ok = 3.ok' and 4.ok = 4.ok' **[Assumption 10]**
  (3 and 4 are labels for some branches of the parallelism, such that i is the label for branch Lift (s_i) ; $d \rightarrow A_i$);

And now we derive other assumptions from **Assumption 8** and **Assumption 9**, called **compldw1** and **compldw2**:
[Assumption 8]
= [Labelled Transition Implication B.17]
$\forall$ w. $c_1 \wedge c_3 \Rightarrow$ (Lift (s_1); A_1) $\sqsubseteq$ (Lift (s_3); $d^* w_1 \rightarrow A_3$) **[Assumption compldw1]**
= w is unviversally quantified, so we abstract it]
$c_1 \wedge c_3 \Rightarrow$ (Lift (s_1); A_1) $\sqsubseteq$ (Lift (s_3); $d^* w_1 \rightarrow A_3$) **[Assumption compldw1]**
[Assumption 9]
= [Labelled Transition Implication B.17]
$c_1 \wedge c_3 \Rightarrow$ (Lift (s_1); A_1) $\sqsubseteq$ (Lift (s_3); $d^* w_1 \rightarrow A_3$) **[Assumption compldw2]**
Now we will start proving.

Proof

$(c_1 \wedge c_2 \mid s \models (c_1 \mid \text{loc } s_1 \bullet A_1) \,[\![x_1 \mid cs \mid x_2]\!]\, (c_2 \mid \text{loc } s_2 \bullet A_2))$
$\xrightarrow{d|w_2}$
$(c_3 \wedge c_4 \wedge w_1 = w_2 \mid s \models ((\text{loc } s_3 \bullet A_3) \,[\![x_1 \mid cs \mid x_2]\!]\, (\text{loc } s_4 \bullet A_4)))$

= [Definition of Labelled Transition]

$$\forall\, w \,.\, c_1 \wedge c_2 \wedge c_3 \wedge c_4 \wedge w_1 = w_2 \Rightarrow \left(\begin{array}{l} (Lift(s);\ (locs_1 \bullet A_1) \,[\![x_1 \mid cs \mid x_2]\!]\, (c_2 \mid locs_2 \bullet A_2)) \\ \sqsubseteq \\ \left(\begin{array}{l} (Lift(s);\ d \mid w_2 \rightarrow ((locs_3 \bullet A_3) \,[\![x_1 \mid cs \mid x_2]\!]\, (locs_4 \bullet A_4))) \\ \Box \\ (Lift(s);\ (locs_1 \bullet A_1) \,[\![x_1 \mid cs \mid x_2]\!]\, (c_2 \mid locs_2 \bullet A_2)) \end{array} \right) \end{array} \right)$$

= [w is universally quantified, so we abstract it]

$$c_1 \wedge c_2 \wedge c_3 \wedge c_4 \wedge w_1 = w_2 \Rightarrow \left(\begin{array}{l} (Lift(s);\ (locs_1 \bullet A_1) \,[\![x_1 \mid cs \mid x_2]\!]\, (c_2 \mid locs_2 \bullet A_2)) \\ \sqsubseteq \\ \left(\begin{array}{l} (Lift(s);\ d \mid w_2 \rightarrow ((locs_3 \bullet A_3) \,[\![x_1 \mid cs \mid x_2]\!]\, (locs_4 \bullet A_4))) \\ \Box \\ (Lift(s);\ (locs_1 \bullet A_1) \,[\![x_1 \mid cs \mid x_2]\!]\, (c_2 \mid locs_2 \bullet A_2)) \end{array} \right) \end{array} \right)$$

= [Communication Parallelism Distribution B.2.7 (having that $\{d\} \in$ CS, as it is Par. Synchronised)]

$$c_1 \wedge c_2 \wedge c_3 \wedge c_4 \wedge w_1 = w_2 \Rightarrow \left(\begin{array}{l} (Lift(s);\ (locs_1 \bullet A_1) \,[\![x_1 \mid cs \mid x_2]\!]\, (c_2 \mid locs_2 \bullet A_2)) \\ \sqsubseteq \\ \left(\begin{array}{l} (Lift(s);\ (d \mid w_2 \rightarrow (locs_3 \bullet A_3) \,[\![x_1 \mid cs \mid x_2]\!]\, d \mid w_2 \rightarrow (locs_4 \bullet A_4))) \\ \Box \\ (Lift(s);\ (locs_1 \bullet A_1) \,[\![x_1 \mid cs \mid x_2]\!]\, (c_2 \mid locs_2 \bullet A_2)) \end{array} \right) \end{array} \right)$$

= [Definition of Loc B.29 6x]

$$c_1 \wedge c_2 \wedge c_3 \wedge c_4 \wedge w_1 = w_2 \Rightarrow \left(\begin{array}{l} (Lift(s);\ (Lift(s_1);\ A_1) \,[\![x_1 \mid cs \mid x_2]\!]\, (c_2 \mid Lift(s_2);\ A_2)) \\ \sqsubseteq \\ \left(\begin{array}{l} (Lift(s);\ (d \mid w_2 \rightarrow (Lift(s_3);\ A_3) \,[\![x_1 \mid cs \mid x_2]\!]\, d \mid w_2 \rightarrow (Lift(s_4);\ A_4))) \\ \Box \\ (Lift(s);\ (Lift(s_1);\ A_1) \,[\![x_1 \mid cs \mid x_2]\!]\, (c_2 \mid Lift(s_2);\ A_2)) \end{array} \right) \end{array} \right)$$

= [Lift Shift 7 (having assumption 10)]

$c_1 \wedge c_2 \wedge c_3 \wedge c_4 \wedge w_1 = w_2 \Rightarrow$

$$\left(\begin{array}{l}(Lift(s);\ (Lift(s_1);\ A_1)\ [\![x_1 \mid cs \mid x_2]\!]\ (c_2 \mid Lift(s_2);\ A_2)) \\ \sqsubseteq \\ \left(\begin{array}{l}(Lift(s);\ ((Lift(s_3);\ d \mid w_2 \to A_3)\ [\![x_1 \mid cs \mid x_2]\!]\ (Lift(s_4);\ d \mid w_2 \to A_4))) \\ \Box \\ (Lift(s);\ (Lift(s_1);\ A_1)\ [\![x_1 \mid cs \mid x_2]\!]\ (c_2 \mid Lift(s_2);\ A_2))\end{array}\right)\end{array}\right)$$

= [If $w_1 \neq w_2$, the antecedent is false, thus the whole expression is true. If $w_1 = w_2$ is true, w_2 can be replaced by w_1]

$c_1 \wedge c_2 \wedge c_3 \wedge c_4 \wedge w_1 = w_2 \Rightarrow$

$$\left(\begin{array}{l}(Lift(s);\ (Lift(s_1);\ A_1)\ [\![x_1 \mid cs \mid x_2]\!]\ (c_2 \mid Lift(s_2);\ A_2)) \\ \sqsubseteq \\ \left(\begin{array}{l}(Lift(s);\ ((Lift(s_3);\ d \mid w_1 \to A_3)\ [\![x_1 \mid cs \mid x_2]\!]\ (Lift(s_4);\ d \mid w_2 \to A_4))) \\ \Box \\ (Lift(s);\ (Lift(s_1);\ A_1)\ [\![x_1 \mid cs \mid x_2]\!]\ (c_2 \mid Lift(s_2);\ A_2))\end{array}\right)\end{array}\right)$$

= [External Choice Refinement Implication 15]

$c_1 \wedge c_2 \wedge c_3 \wedge c_4 \wedge w_1 = w_2 \Rightarrow$

$$\left(\begin{array}{l}(Lift(s);\ (Lift(s_1);\ A_1)\ [\![x_1 \mid cs \mid x_2]\!]\ (c_2 \mid Lift(s_2);\ A_2)) \\ \sqsubseteq \\ \left(\ (Lift(s);\ ((Lift(s_3);\ d \mid w_1 \to A_3)\ [\![x_1 \mid cs \mid x_2]\!]\ (Lift(s_4);\ d \mid w_2 \to A_4)))\ \right)\end{array}\right)$$

= [Assumptions complw1 and complw2 with Monotonicity of Refinement for Parallelism B.2.21]

$c_1 \wedge c_2 \wedge c_3 \wedge c_4 \wedge w_1 = w_2 \Rightarrow$ true

= [Predicate Calculus]

true

□

B.3.9 Hiding

Attached Rule 22.

Hiding Internal:

$$\frac{(c1 \mid s1 \models A) \xrightarrow{l} (c2 \mid s2 \models B) \quad l \in S}{(c1 \mid s1 \models A \setminus S) \xrightarrow{\tau} (c2 \mid s2 \models B \setminus S)}$$

For proving the above rule, we will also assume that (A \ S) and (B \ S) are divergence-free (ok and ok'), and we will call this assumption as **[Assumption Div]**.

On the rule, as it has a labelled transition from program text A to program text B, then A = l → B **[We will call it Assumption 3]**.

[Assumption 1]
(c1 | s1 ⊨ A) $\xrightarrow{l}$ (c2 | s2 ⊨ B)
= [Definition of Labelled Transition B.26]
∀w . c1 ∧ c2 ⇒ Lift (s1) ; A ⊑ (Lift (s2) ; l → B) □ Lift (s1) ; A
= [w is universally quantified, so we abstract it]
c1 ∧ c2 ⇒ Lift (s1) ; A ⊑ (Lift (s2) ; l → B) □ Lift (s1) ; A
= [External Choice Refinement Implication 15]
c1 ∧ c2 ⇒ Lift (s1) ; A ⊑ (Lift (s2) ; l → B)
= [For c1 and c2 being true, we have as follows (if either c1 or c2 is false, the antecedent is false, thus the whole expression is true)]
true ⇒ Lift (s1) ; A ⊑ (Lift (s2) ; l → B)
= [Predicate Calculus]
Lift (s1) ; A ⊑ (Lift (s2) ; l → B)
= [Assumption 3]
Lift (s1) ; l → B ⊑ (Lift (s2) ; l → B)
⇒ **[Hiding Monotonic B.2.9]**
(Lift (s1) ; l → B) \ S ⊑ (Lift (s2) ; l → B) \ S
= [Sequence Skip B.2.2]
(Lift (s1) ; l → Skip ; B) \ S ⊑ (Lift (s2) ; l → Skip ; B) \ S
= [Hidden Event Sequenced By Skip 16 (having ok and ok' from Assumption Div)]
(Lift (s1) ; Skip ; B) \ S ⊑ (Lift (s2) ; Skip ; B) \ S
= [Sequence Skip B.2.2]
(Lift (s1) ; B) \ S ⊑ (Lift (s2) ; B) \ S **[Assumption 4]**

Expression to be proved:

$\forall w \,.\, c1 \wedge c2 \Rightarrow \text{Lift}\ (s1)\ ;\ A \setminus S \sqsubseteq \text{Lift}\ (s2)\ ;\ B \setminus S$

Proof
$(c1 \mid s1 \models A \setminus S) \xrightarrow{\tau} (c2 \mid s2 \models B \setminus S)$
= [Definition of Silent Transition B.25]
$\forall w \,.\, c1 \wedge c2 \Rightarrow \text{Lift}\ (s1)\ ;\ A \setminus S \sqsubseteq \text{Lift}\ (s2)\ ;\ B \setminus S$
= [w is universally quantified, so we abstract it]
$c1 \wedge c2 \Rightarrow \text{Lift}\ (s1)\ ;\ A \setminus S \sqsubseteq \text{Lift}\ (s2)\ ;\ B \setminus S$
= [By Assumption 3]
$c1 \wedge c2 \Rightarrow (\text{Lift}\ (s1)\ ;\ (l \rightarrow B)) \setminus S \sqsubseteq (\text{Lift}\ (s2)\ ;\ B) \setminus S$
= [Hiding Sequence Distributive B.2.12]
$c1 \wedge c2 \Rightarrow (\text{Lift}\ (s1)\ ;\ (l \rightarrow B)) \setminus S \sqsubseteq (\text{Lift}\ (s2)\ ;\ B) \setminus S$
= [Skip A Equals A B.2.2]
$c1 \wedge c2 \Rightarrow (\text{Lift}\ (s1)\ ;\ (l \rightarrow \text{Skip}\ ;\ B)) \setminus S \sqsubseteq (\text{Lift}\ (s2)\ ;\ B) \setminus S$
= [Hiding Sequence Distributive B.2.12]
$c1 \wedge c2 \Rightarrow (\text{Lift}\ (s1)\ ;\ ((l \rightarrow \text{Skip}) \setminus S\ ;\ B \setminus S)) \sqsubseteq (\text{Lift}\ (s2)\ ;\ B) \setminus S$
= [Hidden Event Sequenced by Skip 16]
$c1 \wedge c2 \Rightarrow (\text{Lift}\ (s1)\ ;\ (\text{Skip}\ ;\ B \setminus S)) \sqsubseteq (\text{Lift}\ (s2)\ ;\ B) \setminus S$
= [Skip A Equals A B.2.2]
$c1 \wedge c2 \Rightarrow (\text{Lift}\ (s1)\ ;\ (B \setminus S)) \sqsubseteq (\text{Lift}\ (s2)\ ;\ B) \setminus S$
= [Hiding Identity B.2.8 (as Lift (s1) is a sequence of assignments, it does not contain any channel)]
$c1 \wedge c2 \Rightarrow (\text{Lift}\ (s1) \setminus S\ ;\ (B \setminus S)) \sqsubseteq (\text{Lift}\ (s2)\ ;\ B) \setminus S$
= [Assume c1 $\wedge$ c2 from now on (when c1 $\wedge$ c2 is false, then the whole expression is true)]
$(\text{Lift}\ (s1) \setminus S\ ;\ (B \setminus S)) \sqsubseteq (\text{Lift}\ (s2)\ ;\ B) \setminus S$
= [Hiding Sequence Distributive B.2.12]
$(\text{Lift}\ (s1)\ ;\ B) \setminus S \sqsubseteq (\text{Lift}\ (s2)\ ;\ B) \setminus S$
= [Assumption 4]
true
□

Attached Rule 23.

***Hiding Visible**:*

$$\frac{(c1 \mid s1 \models A) \xrightarrow{l} (c2 \mid s2 \models B) \qquad l \notin S}{(c1 \mid s1 \models A \setminus S) \xrightarrow{l} (c2 \mid s2 \models B \setminus S)}$$

Expression to be proved:

$\forall$w. c1$\wedge$c2 $\Rightarrow$ Lift (s1); A\S $\sqsubseteq$ (Lift (s2); c.w1 $\rightarrow$ B\S) $\Box$ Lift (s1); A\S

Proof

$\forall$w . c1 $\wedge$ c2 $\Rightarrow$

Lift (s1) ; A $\setminus$ S $\sqsubseteq$ (Lift (s2) ; c.w1 $\rightarrow$ B $\setminus$ S) $\Box$ Lift (s1) ; A $\setminus$ S

= [w is universally quantified, so we abstract it]

c1 $\wedge$ c2 $\Rightarrow$

Lift (s1) ; A $\setminus$ S $\sqsubseteq$ (Lift (s2) ; c.w1 $\rightarrow$ B $\setminus$ S) $\Box$ Lift (s1) ; A $\setminus$ S

= [No channels are used on Lift (s), so Lift (s) = Lift (s) $\setminus$ S]

c1 $\wedge$ c2 $\Rightarrow$

Lift (s1) $\setminus$ S ; A $\setminus$ S $\sqsubseteq$ (Lift (s2) $\setminus$ S ; c.w1 $\rightarrow$ B $\setminus$ S) $\Box$ Lift (s1) $\setminus$ S ; A $\setminus$ S

= [Hiding Sequence Distributive B.2.12]

c1 $\wedge$ c2 $\Rightarrow$

(Lift (s1) ; A) $\setminus$ S $\sqsubseteq$ ((Lift (s2) ; c.w1 $\rightarrow$ B) $\setminus$ S) $\Box$ (Lift (s1) ; A) $\setminus$ S

= [Hiding External Choice Distributive B.2.10]

c1 $\wedge$ c2 $\Rightarrow$

(Lift (s1) ; A) $\setminus$ S $\sqsubseteq$ ((Lift (s2) ; c.w1 $\rightarrow$ B) $\Box$ (Lift (s1) ; A)) $\setminus$ S

= [Assumption and Hiding Monotonic B.2.9]

c1 $\wedge$ c2 $\Rightarrow$ true

= [Predicate Calculus]

true $\Box$

Attached Rule 24.

***Hiding Skip**:*

$(c \mid s \models Skip \setminus S) \xrightarrow{\tau} (c \mid s \models Skip)$

Proof We start this proof by infering an assumption from function UsedC (B.12):

(UsedC (Skip) = { }) $\Rightarrow$ (New Assumption) (**S** $\cap$ **UsedC (Skip) =** { })

Now we will continue the proof:

(c | s $\models$ Skip $\setminus$ S) $\xrightarrow{\tau}$ (c | s $\models$ Skip)

= [From New Assumption and Law B.2.8]

$(c \mid s \models \text{Skip}) \xrightarrow{\tau} (c \mid s \models \text{Skip})$

= [Silent Transition between Equivalent nodes 2]

true □

B.3.10 Recursion

Attached Rule 25.

Recursion:

$$\frac{(c \mid s \models A) \xrightarrow{\tau} (c \mid s \models B)\ N = A}{(c \mid s \models N) \xrightarrow{\tau} (c \mid s \models B)}$$

Proof The proof for Recursion is straightforward. We only apply the assumption N = A and then the other assumption to prove:
(c | s $\models$ N) $\xrightarrow{\tau}$ (c | s $\models$ B)
= [Assumption N = A]
(c | s $\models$ A) $\xrightarrow{\tau}$ (c | s $\models$ B)
= [Assumption (c | s $\models$ A) $\xrightarrow{\tau}$ (c | s $\models$ B)]
true □

B.3.11 Call

Attached Rule 26.

Call:

$$\frac{CALL = Content\ (CALL)}{(c \mid s \models CALL) \xrightarrow{\tau} (c \mid s \models Content\ (CALL))}$$

Proof (c | s $\models$ CALL) $\xrightarrow{\tau}$ (c | s $\models$ Content (CALL))
= [Assumption CALL = Content (CALL)]
(c | s $\models$ Content (CALL)) $\xrightarrow{\tau}$ (c | s $\models$ Content (CALL))
= [Silent Transition Between Equivalent Nodes 2]
true □

B.3.12 Iterated Actions

Attached Rule 27.

Iterated Actions:
(c | s $\models$ OP Decl • A) $\xrightarrow{\tau}$ (c | s $\models$ IteratedExpansion (A, Decl, ITOPFLAG))
IteratedExpansion is defined on B.3. ITOPFLAG, as indicated on B.3, is a flag that indicates the operator that is being iterated.

Proof. (c | s $\models$ OP Decl $\bullet$ A) $\xrightarrow{\tau}$ (c | s $\models$ IteratedExpansion (A, Decl, ITOPFLAG))
= [Definition of Iterated Operator B.3]
(c | s $\models$ IteratedExpansion (A, Decl, ITOPFLAG)) $\xrightarrow{\tau}$ (c | s $\models$ IteratedExpansion (A, Decl, ITOPFLAG))
= [Silent Transition Equal Sides 2]
true □

B.3.13 If-Guarded Command

Attached Rule 28.

If-Guarded Command:
Be

$IGC \mathrel{\widehat{=}} \textbf{if } (pred_1) \rightarrow A_1 \,[\!]\, (pred_2) \rightarrow A_2 \,[\!]\, \ldots \,[\!]\, (pred_n) \rightarrow A_n \textbf{ fi}$, *then*

$(c \mid s \models IGC) \xrightarrow{\tau} (c \wedge pred_1 \mid s \models A_1)\ \boldsymbol{T}_1$
$(c \mid s \models IGC) \xrightarrow{\tau} (c \wedge pred_2 \mid s \models A_2)\ \boldsymbol{T}_2$
...
$(c \mid s \models IGC) \xrightarrow{\tau} (c \wedge pred_n \mid s \models A_n)\ \boldsymbol{T}_n$

$(c \mid s \models IGC)$
$\xrightarrow{\tau}$
$(c \wedge (\neg\ pred_1) \wedge (\neg\ pred_2) \wedge \ldots \wedge (\neg\ pred_{n-1}) \wedge (\neg\ pred_n) \mid s \models Chaos)\ \boldsymbol{T}_{Chaos}$

Proof

Proof of $\boldsymbol{T}_1$

Among rules T_i, we will prove only law T_1. The other laws can be proved using similar reasoning.

$(c \mid s \models IGC) \xrightarrow{\tau} (c \wedge pred_1 \mid s \models A_1)$
= [Definition of Silent Transition B.25]
$\forall\ w\ .\ c \wedge c \wedge pred_1 \Rightarrow (Lift\ (s)\ ;\ IGC \sqsubseteq Lift\ (s)\ ;\ A_1)$
= [w is universally quantified, so we abstract it]
$c \wedge c \wedge pred_1 \Rightarrow (Lift\ (s)\ ;\ IGC \sqsubseteq Lift\ (s)\ ;\ A_1)$
= [Monotonicity of Refinement 6: if Lift (s) $\sqsubseteq$ Lift (s) and IGC $\sqsubseteq$ A_1 (by B.12 and (assms) $pred_1$ ($pred_1$ is true because it appears on the left side of the implication))]
$c \wedge c \wedge pred_1 \Rightarrow true$
= [Predicate Calculus]
true

Now we will prove T_{Chaos} rule.

Proof of <T_{Chaos}> rule:

$(c \mid s \models IGC) \xrightarrow{\tau} (c \wedge (\neg\ pred_1) \wedge ... \wedge (\neg\ pred_n) \mid s \models Chaos)$
=
$\forall\ w\ .\ (c \wedge c \wedge (\neg\ pred_1) \wedge ... \wedge (\neg\ pred_n)) \Rightarrow (Lift\ (s)\ ;\ IGC \sqsubseteq Lift\ (s)\ ;\ Chaos)$
= [w is universally quantified, so we abstract it]
$(c \wedge c \wedge (\neg\ pred_1) \wedge ... \wedge (\neg\ pred_n)) \Rightarrow (Lift\ (s)\ ;\ IGC \sqsubseteq Lift\ (s)\ ;\ Chaos)$
= [c is true]
$((\neg\ pred_1) \wedge ... \wedge (\neg\ pred_n)) \Rightarrow (Lift\ (s)\ ;\ IGC \sqsubseteq Lift\ (s)\ ;\ Chaos)$
= [$(\neg$ pred$_1$) $\wedge$ ... $\wedge$ ($\neg$ pred$_n$) $\Rightarrow$ (IGC = Chaos)]
$((\neg\ pred_1) \wedge ... \wedge (\neg\ pred_n)) \Rightarrow (Lift\ (s)\ ;\ Chaos \sqsubseteq Lift\ (s)\ ;\ Chaos)$
= [E $\sqsubseteq$ E]
$((\neg\ pred_1) \wedge ... \wedge (\neg\ pred_n)) \Rightarrow true$
=
true
□

B.3.14 Z Schema

Attached Rule 29.

***Z Schema**:*

$$\frac{c \wedge (s\ ;\ \textbf{\textit{pre}}\ Op)}{(c \mid s \models Op) \xrightarrow{\tau} (c \wedge (s\ ;\ Op\ [w0/v'\,]) \mid s;\ v := w0 \models Skip)}$$

The **pre** operator is defined on B.2.

Proof

$\forall\ w\ .\ c \wedge c \wedge (s\ ;\ Op\ [w0/v'\,]) \Rightarrow ((Lift\ (s)\ ;\ Op) \sqsubseteq (Lift\ (s;\ v := w0)\ ;\ Skip))$

The proof is divided in two steps: we firstly prove the expression for Op = true, and then prove it for Op = false

For Op = true:
$\forall\ w\ .\ c \wedge c \wedge (s\ ;\ true\ [w0/v'\,]) \Rightarrow ((Lift\ (s)\ ;\ true) \sqsubseteq (Lift\ (s;\ v := w0)\ ;\ Skip))$
= [w is universally quantified, so we abstract it]
$c \wedge c \wedge (s\ ;\ true\ [w0/v'\,]) \Rightarrow ((Lift\ (s)\ ;\ true) \sqsubseteq (Lift\ (s;\ v := w0)\ ;\ Skip))$
= [Lift Composition B.2.27]
$c \wedge c \wedge (s\ ;\ true\ [w0/v'\,]) \Rightarrow ((Lift\ (s)\ ;\ true) \sqsubseteq (Lift\ (s);\ Lift\ (v := w0)\ ;\ Skip))$

= [By lemma B.11, (true ⊑ Lift (v := w0) ; Skip). By Ref.Eq.Sides 1, (Lift (s) ⊑ Lift (s)). By both B.11 and 1 and by Monotonicity of refinement, the refinement expression on the right side of the implication is true]

c ∧ c ∧ (s ; true [w0/v']) ⇒ true

= [Predicate Calculus]

true

For Op = false:

∀ w . c ∧ c ∧ (s ; false [w0/v']) ⇒ ((Lift (s) ; false) ⊑ (Lift (s; v := w0) ; Skip))

= [w is universally quantified, so we abstract it]

c ∧ c ∧ (s ; false [w0/v']) ⇒ ((Lift (s) ; false) ⊑ (Lift (s; v := w0) ; Skip))

= [false [w/x] = false]

c ∧ c ∧ (s ; false) ⇒ ((Lift (s) ; false) ⊑ (Lift (s; v := w0) ; Skip))

= [S Sequence False 5]

c ∧ c ∧ false ⇒ ((Lift (s) ; false) ⊑ (Lift (s; v := w0) ; Skip))

= [Predicate Calculus]

false ⇒ ((Lift (s) ; false) ⊑ (Lift (s; v := w0) ; Skip))

= [Predicate Calculus]

true

□

B.3.15 Specification Statements

Attached Rule 30.

Specification Statements (Rule 1):

(c ∣ s ⊨ V : [Pre, Post]) $\xrightarrow{\tau}$ (c ∧ Pre ∣ s ⊨ V [:] [Pre, Post])

The Extra Statement ([:]) operator B.31 has the role of representing the Specification Statement in an intermediate state where its pre-condition is holded but its post-condition was not processed yet. The role of the Extra-statement operator is the same as the **let** B.28 operator on the rules of Variable Block.

Proof

(c ∣ s ⊨ V : [Pre, Post]) $\xrightarrow{\tau}$ (c ∧ Pre ∣ s ⊨ V [:] [Pre, Post])

= [Extra statement (definition B.31)]

(c ∣ s ⊨ V : [Pre, Post]) $\xrightarrow{\tau}$ (c ∧ Pre ∣ s ⊨ V : [Pre, Post])

= [Silent Transition Equal Sides (lemma 2)]

true □

Attached Rule 31.

***Specification Statements (Rule 2)**:*
(c ∣ s ⊨ V : [Pre, Post]) $\xrightarrow{\tau}$ (c ∧ (¬ Pre) ∣ s ⊨ Skip)

Proof
(c ∣ s ⊨ V : [Pre, Post]) $\xrightarrow{\tau}$ (c ∧ (¬ Pre) ∣ s ⊨ Skip)
= [Skip V G True B.2.1]
∀ w . c ∧ c ∧ (¬ Pre) ⇒ Lift (s) ; V : [Pre, Post] ⊑ Lift (s) ; V : [true, true]

From now on, the proof will be made for all 4 different situations of Pre and Post:

Pre = Post = true:

∀ w . c ∧ c ∧ (¬ true) ⇒ Lift (s) ; V : [true, true] ⊑ Lift (s) ; V : [true, true]
= [w is universally quantified, so we abstract it]
c ∧ c ∧ (¬ true) ⇒ Lift (s) ; V : [true, true] ⊑ Lift (s) ; V : [true, true]
= [Predicate Calculus]
false ⇒ true
= [Predicate Calculus]
true

Pre = true, Post = false:

∀ w . c ∧ c ∧ (¬ true) ⇒ Lift (s) ; V : [true, false] ⊑ Lift (s) ; V : [true, true]
= [w is universally quantified, so we abstract it]
c ∧ c ∧ (¬ true) ⇒ (Lift (s) ; V : [true, false] ⊑ Lift (s) ; V : [true, true])
= false ⇒ (Lift (s) ; V : [true, false] ⊑ Lift (s) ; V : [true, true])
= true

Pre = false, Post = true:

(c ∣ s ⊨ V : [false, true]) $\xrightarrow{\tau}$ (c ∧ (¬ false) ∣ s ⊨ Skip)
= [Definition of Silent Transition B.25]

$\forall$ w . c $\wedge$ c $\wedge$ ($\neg$ false) $\Rightarrow$ Lift (s) ; V : [false, true] $\sqsubseteq$ Lift (s) ; V : [true, true]
= [w is universally quantified, so we abstract it]
c $\wedge$ c $\wedge$ ($\neg$ false) $\Rightarrow$ Lift (s) ; V : [false, true] $\sqsubseteq$ Lift (s) ; V : [true, true]
= [Assume c = true]
Lift (s) ; V : [false, true] $\sqsubseteq$ Lift (s) ; V : [true, true]
= [As Lift (s) is refined by itself, so, from Monotonicity of Refinement, the proof from now on will consist only on proving that V : [false, true] $\sqsubseteq$ V : [true, true]]
V : [false, true] $\sqsubseteq$ V : [true, true]
= [Specification Statement Denotational Definition A.6]
R (false $\vdash$ true $\wedge$ $\neg$ wait' $\wedge$ tr' = tr $\wedge$ u' = u) $\sqsubseteq$ R(true $\vdash$ true $\wedge$ $\neg$ wait' $\wedge$ tr' = tr $\wedge$ u' = u)
= [Predicate Calculus]
R ((false $\wedge$ ok)
$\Rightarrow$
((true $\wedge$ $\neg$ wait' $\wedge$ tr' = tr $\wedge$ u' = u) $\wedge$ ok') $\sqsubseteq$ R((true $\wedge$ ok) $\Rightarrow$ (true $\wedge$ $\neg$ wait' $\wedge$ tr' = tr $\wedge$ u' = u $\wedge$ ok'))
= [Predicate Calculus]
R (true)
$\sqsubseteq$ R((true $\wedge$ ok) $\Rightarrow$ (true $\wedge$ $\neg$ wait' $\wedge$ tr' = tr $\wedge$ u' = u $\wedge$ ok'))
= [Predicate Calculus]
R (true)
$\sqsubseteq$ R(($\neg$true $\vee$ $\neg$ok) $\vee$ (true $\wedge$ $\neg$ wait' $\wedge$ tr' = tr $\wedge$ u' = u $\wedge$ ok'))
= [Predicate Calculus]
R (true)
$\sqsubseteq$ R(($\neg$ok) $\vee$ ($\neg$ wait' $\wedge$ tr' = tr $\wedge$ u' = u $\wedge$ ok'))
= [Be x = ($\neg$ok) $\vee$ ($\neg$ wait' $\wedge$ tr' = tr $\wedge$ u' = u $\wedge$ ok')]
R (true) $\sqsubseteq$ R(x)
= [Healthy True Refinement B.6]
true
Pre = Post = false:

(c | s $\models$ V : [Pre,Post])$\xrightarrow{\tau}$(c $\wedge$ ($\neg$Pre) | s $\models$ Skip)
= [Definition of Silent Transition B.25]
$\forall$ w . c $\wedge$ c $\wedge$ ($\neg$ false) $\Rightarrow$ Lift (s) ; V : [false, false] $\sqsubseteq$ Lift (s) ; V : [true, true]
= [w is universally quantified, so we abstract it]
c $\wedge$ c $\wedge$ ($\neg$ false) $\Rightarrow$ Lift (s) ; V : [false, false] $\sqsubseteq$ Lift (s) ; V : [true, true]
= [Predicate Calculus]

Lift (s) ; V : [false, false] $\sqsubseteq$ Lift (s) ; V : [true, true]
= **[Monotonicity of Refinement 6]**
V : [false, false] $\sqsubseteq$ V : [true, true]
= **[Predicate Calculus]**
R (false $\vdash$ false $\wedge$ $\neg$ wait' $\wedge$ tr' = tr $\wedge$ u' = u)
$\sqsubseteq$ R(true $\vdash$ true $\wedge$ $\neg$ wait' $\wedge$ tr' = tr $\wedge$ u' = u)
= **[Predicate Calculus]**
R (false $\vdash$ false) $\sqsubseteq$ R(true $\vdash$ $\neg$ wait' $\wedge$ tr' = tr $\wedge$ u' = u)
= **[Predicate Calculus]**
R (false $\wedge$ ok $\Rightarrow$ false $\wedge$ ok') $\sqsubseteq$ R(true $\vdash$ $\neg$ wait' $\wedge$ tr' = tr $\wedge$ u' = u)
= **[Predicate Calculus]**
R (false $\Rightarrow$ false) $\sqsubseteq$ R(true $\vdash$ $\neg$ wait' $\wedge$ tr' = tr $\wedge$ u' = u)
= **[Predicate Calculus]**
R (true) $\sqsubseteq$ R(true $\vdash$ $\neg$ wait' $\wedge$ tr' = tr $\wedge$ u' = u)
= **[Healthy True Refinement B.6]**
true
□

Attached Rule 32.

Specification Statements (Rule 3)*:*
(c $\wedge$ Pre | s $\models$ V [:] [Pre, Post])
$\xrightarrow{\tau}$
(c $\wedge$ Pre $\wedge$ Post $\wedge$ (α(P) - V)' = (α(P) - V)) | s $\models$ Skip)

Proof

(c $\wedge$ Pre | s $\models$ V [:] [Pre, Post])
$\xrightarrow{\tau}$
(c $\wedge$ Pre $\wedge$ (Post $\wedge$ (α(P) - V)' = (α(P) - V)) | s $\models$ Skip)
= **[Definition of Silent Transition B.25]**
$\forall$ w . (c $\wedge$ Pre $\wedge$ c $\wedge$ Pre $\wedge$ Post $\wedge$ (α(P) - V)' = (α(P) - V))
$\Rightarrow$ Lift (s) ; V [:] [Pre, Post] $\sqsubseteq$ Lift (s) ; Skip

We will prove the above expression checking the following complementary situations:
(Pre = false $\vee$ Post = false) and (Pre = true $\wedge$ Post = true)

<u>Pre = false $\vee$ Post = false</u>

In this case, the antecedent of the implication is false:

$\forall$ w . (c $\wedge$ false $\wedge$ c $\wedge$ Pre $\wedge$ Post $\wedge$ (α(P) - V)' = (α(P) - V))
$\Rightarrow$ Lift (s) ; V [:] [Pre, Post] $\sqsubseteq$ Lift (s) ; Skip
= [w is universally quantified, so we abstract it]
(c $\wedge$ false $\wedge$ c $\wedge$ Pre $\wedge$ Post $\wedge$ (α(P) - V)' = (α(P) - V))
$\Rightarrow$ Lift (s) ; V [:] [Pre, Post] $\sqsubseteq$ Lift (s) ; Skip
= [Predicate Calculus]
(false $\Rightarrow$ Lift (s) ; V [:] [Pre, Post] $\sqsubseteq$ Lift (s) ; Skip
= [Predicate Calculus]
true

Pre = true $\wedge$ Post = true
$\forall$ w . (c $\wedge$ true $\wedge$ c $\wedge$ true $\wedge$ true $\wedge$ (α(P) - V)' = (α(P) - V))
$\Rightarrow$ Lift (s) ; V [:] [true, true] $\sqsubseteq$ Lift (s) ; Skip
= [w is universally quantified, so we abstract it]
(c $\wedge$ true $\wedge$ c $\wedge$ true $\wedge$ true $\wedge$ (α(P) - V)' = (α(P) - V))
$\Rightarrow$ Lift (s) ; V [:] [true, true] $\sqsubseteq$ Lift (s) ; Skip
= [Predicate Calculus]
(c $\wedge$ true $\wedge$ c $\wedge$ true $\wedge$ true $\wedge$ (α(P) - V)' = (α(P) - V))
$\Rightarrow$ Lift (s) ; V [:] [true, true] $\sqsubseteq$ Lift (s) ; V : [true, true]
= [Predicate Calculus]
(c $\wedge$ true $\wedge$ c $\wedge$ true $\wedge$ true $\wedge$ (α(P) - V)' = (α(P) - V))
$\Rightarrow$ Lift (s) ; V : [true, true] $\sqsubseteq$ Lift (s) ; V : [true, true]
= [Refinement Equal Sides 1]
(c $\wedge$ true $\wedge$ c $\wedge$ true $\wedge$ true $\wedge$ (α(P) - V)' = (α(P) - V))
$\Rightarrow$ true
= [Predicate Calculus]
= true
$\square$

B.3.16 Basic Process

Attached Rule 33.

Basic Process Begin:

$$\frac{(c1 \mid s1 \models A1) \xrightarrow{lp} (c2 \mid s2 \models A2)}{\begin{array}{l}(c1 \mid s1 \mid \textbf{\textit{begin state}}\ [Vars\text{-}decl \mid inv] \bullet A1\ \textbf{\textit{end}}) \\ \xrightarrow{lp} (c2 \mid s2 \mid \textbf{\textit{begin state}}\ [Vars\text{-}decl \mid inv] \bullet A2\ \textbf{\textit{end}})\end{array}}$$

Assumption:

$(c1 \mid s1 \models A1) \xrightarrow{lp} (c2 \mid s2 \models A2)$

= [Definition of Labelled Transition B.26]

$\forall w \,.\, c1 \wedge c2 \Rightarrow$

$$\left(\begin{array}{l} Lift(s1);\ A1 \\ \sqsubseteq Lift(s2);\ lp \rightarrow A1 \ \Box\ Lift(s1);\ A1 \end{array}\right)$$

= [Labelled Transition Implication B.17]

$\forall\, w \,.\, (\text{Lift } (s1);\ A1) \sqsubseteq \forall\, w \,.\, (\text{Lift } (s3);\ lp \rightarrow A3)$ **[Assumption 2]**

Proof

$(c1 \mid s1 \mid$ **begin state** $[Vars\text{-}decl \mid inv] \bullet A1$ **end**$)$

$\xrightarrow{lp} (c2 \mid s2 \mid$ **begin state** $[Vars\text{-}decl \mid inv] \bullet A2$ **end**$)$

= [Definition of Labelled Transition B.26]

$\forall w.c1 \wedge c2 \Rightarrow$

$Lift(s1);\ (\textbf{begin state}[Vars - decl \mid inv] \bullet A1 \textbf{end})$

$$\sqsubseteq$$

$$\left(\begin{array}{l} Lift(s2);\ lp \rightarrow \textbf{begin state}[Vars - decl \mid inv] \bullet A1\ \textbf{end} \\ \Box\ Lift(s1);\ \textbf{begin state}[Vars - decl \mid inv] \bullet A1\ \textbf{end} \end{array}\right)$$

= [Basic Process Denotational Meaning B.2.15]

$\forall w.c1 \wedge c2 \Rightarrow$

$Lift(s1);\ Vars - Decl \bullet A1$

$$\sqsubseteq$$

$$\left(\begin{array}{l} Lift(s2);\ lp \rightarrow Vars - Decl \bullet A1 \\ \Box\ Lift(s1);\ Vars - Decl \bullet A1 \end{array}\right)$$

= [w is universally quantified, so we abstract it]

$c1 \wedge c2 \Rightarrow$

$Lift(s1);\ Vars - Decl \bullet A1$

$\sqsubseteq$

$$\left(\begin{array}{l} Lift(s2);\ lp \rightarrow Vars - Decl \bullet A1 \\ \Box\ Lift(s1);\ Vars - Decl \bullet A1 \end{array}\right)$$

= [External Choice Idempotence B.2.24]

$c1 \wedge c2 \Rightarrow$

$(Lift(s1);\ Vars - Decl \bullet A1)\ \Box\ (Lift(s1);\ Vars - Decl \bullet A1)$

$\sqsubseteq$

$$\left(\begin{array}{l} Lift(s2);\ lp \rightarrow Vars - Decl \bullet A1 \\ \Box\ Lift(s1);\ Vars - Decl \bullet A1 \end{array}\right)$$

= [Assumption 2 + External Choice Monotonic 14]

$c1 \wedge c2 \Rightarrow true$

= [Predicate Calculus]

true □

Attached Rule 34.

***Basic Process Reduction**:*

Based on B.2.16 and B.2.17 from the refinement calculus of Circus:

$$\frac{c \qquad \alpha(STA) \cap \alpha(STB) = \emptyset}{(c \models (\textbf{begin state}\ STA\ PARS\text{-}A \bullet A\ \textbf{end})\ OP\ (\textbf{begin state}\ STB\ PARS\text{-}B \bullet B\ \textbf{end}))}$$

$\rightsquigarrow$

$(c \mid s \models \textbf{begin state}\ (STA \wedge STB)\ (PARS\text{-}A \wedge_{\Xi} STB)\ (PARS\text{-}B \wedge_{\Xi} STA) \bullet A\ OP\ B\ \textbf{end})$

Where $OP \in \{ \sqcap, \Box, ;, |||, [\![CS]\!], [\![\alpha(A) \mid CS \mid \alpha(B)]\!] \}$

As Basic Process Reduction is an abstract rule applied for a set of binary operators that appear as processes and as actions (Internal Choice, External Choice, Parallelism, Sequence, Interleaving), the proof involves the application of different refinement laws, depending on the operator. But the sequence of proof sub-goals is the same. As the proof is based on the application of rules B.2.16 and B.2.17. Be

B = $(\textbf{begin state}\ STA\ PARS - A \bullet A\ \textbf{end})\ OP\ (\textbf{begin state}\ STB\ PARS - B \bullet B\ \textbf{end})$, and

P = $\textbf{begin state}\ (STA \wedge STB)\ (PARS - A \wedge_{\Xi} STB)\ (PARS - B \wedge_{\Xi} STA) \bullet A\ OP\ B\ \textbf{end})$

Proof

$$\left(\begin{array}{l} (c \models (\textbf{begin state}\ STA\ PARS - A \bullet A\ \textbf{end})\ OP\ (\textbf{begin state}\ STB\ PARS - B \bullet B\ \textbf{end})) \\ \rightsquigarrow \\ (c \mid s \models \textbf{begin state}\ (STA \wedge STB)\ (PARS - A \wedge_{\Xi} STB)\ (PARS - B \wedge_{\Xi} STA) \bullet A\ OP\ B\ \textbf{end}) \end{array}\right)$$

= [Sigma Equal Basic Process (having, from def.B.2.16 and B.2.17, that B=P]
true
□

B.3.17 Compound Process

Attached Rule 35.

Compound Process Left:

$$\frac{(c_1 \models P_1) \rightsquigarrow (c_2 \models P_3)}{(c_1 \models P_1 \ OP \ P_2) \rightsquigarrow (c_2 \models P_3 \ OP \ P_2)}$$

Where $OP \in \{ \sqcap, \square, ; , |||, [\![CS]\!] \}$

The proof will be divided on the following steps:

- Derivation of the assumption on other minor assumptions;
- Creation of a Lemma to synthesize the proof (the lemmas are referenced on the tactics and have their proofs made on appendix B.2.2);
- Division of the rule on a Sub-rule for each operator OP (OP $\in \{ \sqcap, \square, ; , |||, [\![CS]\!] \}$), and proof for each Sub-rule;

To reduce verbose, we create the abbreviation $gA(s)$ such that
$gA(s) = getAssignments(s)$

Assumptions B.19. .
[Assumption 1]
$(c_1 \models P_1) \rightsquigarrow (c_2 \models P_3)$ =
[Definition of Syntactic Transition having w universally quantified B.27]
$c_1 \wedge c_2 \Rightarrow$
$((Lift\ (gA\ (P_1))\ ;\ P_1 \sqsubseteq Lift\ (gA\ (P_3))\ ;\ P_3)) \wedge Lift\ (gA\ (P_1)) = Lift\ (gA\ (P_3))$

$((Lift\ (gA\ (P_1))\ ;\ P_1 \sqsubseteq Lift\ (gA\ (P_3))\ ;\ P_3)) \wedge Lift\ (gA\ (P_1)) = Lift\ (gA\ (P_3)) \Rightarrow$
[Assumption 2] $Lift\ (gA\ (P_1)) = Lift\ (gA\ (P_3))$
$((Lift\ (gA\ (P_1))\ ;\ P_1 \sqsubseteq Lift\ (gA\ (P_3))\ ;\ P_3)) \wedge Lift\ (gA\ (P_1)) = Lift\ (gA\ (P_3)) \Rightarrow$
[Assumption 3] $Lift\ (gA\ (P_1))\ ;\ P_1 \sqsubseteq Lift\ (gA\ (P_3))\ ;\ P_3$
As ***[Assumption 2]*** *and* ***[Assumption 3]***
then
[Assumption 4] $Lift\ (gA\ (P_3))\ ;\ P_1 \sqsubseteq Lift\ (gA\ (P_3))\ ;\ P_3$
As ***[Assumption 2]*** *and* ***[Assumption 4]***
[Assumption 5] $P_1 \sqsubseteq P_3$

Sub-Rule 1.

Internal Choice Compound Process Left:

$$\frac{(c_1 \models P_1) \rightsquigarrow (c_2 \models P_3) \qquad OP = \sqcap}{(c_1 \models P_1 \; OP \; P_2) \rightsquigarrow (c_2 \models P_3 \; OP \; P_2)}$$

Proof

$(c_1 \models P_1 \text{ OP } P_2) \rightsquigarrow (c_2 \models P_3 \text{ OP } P_2)$

= [Lemma Compound Process B.9]

$c_1 \wedge c_2 \Rightarrow$

$(\text{Lift } (\text{gA } (P_1)) \wedge \text{Lift } (\text{gA } (P_2)) \; ; P_1 \text{ OP } P_2 \sqsubseteq (\text{Lift } (\text{gA } (P_1)) \wedge \text{Lift } (\text{gA } (P_2)) \; ; P_3 \text{ OP } P_2))$

= [Be L = (Lift (gA (P_1)) $\wedge$ Lift (gA (P_2))]

$c_1 \wedge c_2 \Rightarrow$

$L \; ; P_1 \text{ OP } P_2 \sqsubseteq L \; ; P_3 \text{ OP } P_2$

= [Assumption OP = $\sqcap$]

$c_1 \wedge c_2 \Rightarrow$

$L \; ; P_1 \sqcap P_2 \sqsubseteq (L \; ; P_3 \sqcap P_2)$

= [$P_1 \sqsubseteq P_3$ (Assumption 5) $\wedge$ $P_2 \sqsubseteq P_2$, so $P_1 \sqcap P_2 \sqsubseteq P_3 \sqcap P_2$ (B.2.20). $L \sqsubseteq L \wedge P_1 \sqcap P_2 \sqsubseteq P_3 \sqcap P_2$, then (Monotonicity of Refinement 6) $L \; ; P_1 \sqcap P_2 \sqsubseteq L \; ; P_3 \sqcap P_2$]

$c_1 \wedge c_2 \Rightarrow \text{true}$

= [Predicate Calculus: (P $\Rightarrow$ true) = true]

true

= true □

Sub-Rule 2.

External Choice Compound Process Left:

$$\frac{(c_1 \models P_1) \rightsquigarrow (c_2 \models P_3) \qquad OP = \square}{(c_1 \models P_1 \; OP \; P_2) \rightsquigarrow (c_2 \models P_3 \; OP \; P_2)}$$

Proof

$(c_1 \models P_1 \text{ OP } P_2) \rightsquigarrow (c_2 \models P_3 \text{ OP } P_2)$

= [Lemma Compound Process B.9]

$c_1 \wedge c_2 \Rightarrow$

$(\text{Lift } (\text{gA } (P_1)) \wedge \text{Lift } (\text{gA } (P_2)) \; ; P_1 \text{ OP } P_2 \sqsubseteq (\text{Lift } (\text{gA } (P_1)) \wedge \text{Lift } (\text{gA } (P_2)) \; ; P_3 \text{ OP } P_2))$

= [Be L = (Lift (gA (P_1)) $\wedge$ Lift (gA (P_2))]

$c_1 \wedge c_2 \Rightarrow$

$L \; ; P_1 \text{ OP } P_2 \sqsubseteq L \; ; P_3 \text{ OP } P_2$

= [Assumption OP = □]

$c_1 \wedge c_2 \Rightarrow$

$L ; P_1 \Box P_2 \sqsubseteq (L ; P_3 \Box P_2)$
= [$P_1 \sqsubseteq P_3$ (Assumption 5) $\wedge$ $P_2 \sqsubseteq P_2$, so $P_1 \Box P_2 \sqsubseteq P_3 \Box P_2$ (B.2.22). $L \sqsubseteq L \wedge P_1 \Box P_2 \sqsubseteq P_3 \Box P_2$, then (Monotonicity of Refinement 6) $L ; P_1 \Box P_2 \sqsubseteq L ; P_3 \Box P_2$]
$c_1 \wedge c_2 \Rightarrow$ true
= (P $\Rightarrow$ true) = true
true
= true □

Sub-Rule 3.

Sequence Compound Process Left:

$$\frac{(c_1 \models P_1) \rightsquigarrow (c_2 \models P_3) \qquad OP = ;}{(c_1 \models P_1\ OP\ P_2) \rightsquigarrow (c_2 \models P_3\ OP\ P_2)}$$

Proof

$(c_1 \models P_1\ OP\ P_2) \rightsquigarrow (c_2 \models P_3\ OP\ P_2)$
= [Lemma Compound Process B.9]
$c_1 \wedge c_2 \Rightarrow$
$(Lift\ (gA\ (P_1)) \wedge Lift\ (gA\ (P_2)) ; P_1\ OP\ P_2 \sqsubseteq (Lift\ (gA\ (P_1)) \wedge Lift\ (gA\ (P_2)) ; P_3\ OP\ P_2))$
= [Be L = (Lift (gA (P_1)) $\wedge$ Lift (gA (P_2))]
$c_1 \wedge c_2 \Rightarrow$
$L ; P_1\ OP\ P_2 \sqsubseteq L ; P_3\ OP\ P_2$
= [Assumption OP = ;]
$c_1 \wedge c_2 \Rightarrow$
$L ; P_1 ; P_2 \sqsubseteq (L ; P_3 ; P_2)$
= [$P_1 \sqsubseteq P_3$ (Assumption 5) $\wedge$ $P_2 \sqsubseteq P_2$, so $P_1 ; P_2 \sqsubseteq P_3 ; P_2$ (6)]. $L \sqsubseteq L \wedge P_1 ; P_2 \sqsubseteq P_3 ; P_2$, then (Monotonicity of Refinement 6) $L ; P_1 ; P_2 \sqsubseteq L ; P_3 ; P_2$]
$c_1 \wedge c_2 \Rightarrow$ true
= (P $\Rightarrow$ true) = true
true
= true □

Sub-Rule 4.

Parallelism Compound Process Left:

$$\frac{(c_1 \models P_1) \rightsquigarrow (c_2 \models P_3) \qquad OP = [\![\, CS \,]\!]}{(c_1 \models P_1\ OP\ P_2) \rightsquigarrow (c_2 \models P_3\ OP\ P_2)}$$

Proof

$(c_1 \models P_1\ OP\ P_2) \rightsquigarrow (c_2 \models P_3\ OP\ P_2)$

= [Lemma Compound Process B.9]

$c_1 \wedge c_2 \Rightarrow$

$(\text{Lift } (gA\ (P_1)) \wedge \text{Lift } (gA\ (P_2)) \ ;\ P_1 \ OP\ P_2 \sqsubseteq (\text{Lift } (gA\ (P_1)) \wedge \text{Lift } (gA\ (P_2)) \ ;\ P_3 \ OP\ P_2))$

= [Be L = (Lift (gA (P_1)) $\wedge$ Lift (gA (P_2))]

$c_1 \wedge c_2 \Rightarrow$

$L \ ; P_1 \ OP\ P_2 \sqsubseteq L \ ; P_3 \ OP\ P_2$

= [Assumption OP = $[\![$ CS $]\!]$]

$c_1 \wedge c_2 \Rightarrow$

$L \ ; P_1 \ [\![CS]\!] \ P_2 \sqsubseteq (L \ ; P_3 \ [\![CS]\!] \ P_2)$

= [$P_1 \sqsubseteq P_3$ (Assumption 5) $\wedge$ $P_2 \sqsubseteq P_2$, so P_1 $[\![$ CS $]\!]$ $P_2 \sqsubseteq P_3$ $[\![$ CS $]\!]$ P_2 (B.2.21). L $\sqsubseteq$ L $\wedge$ P_1 $[\![$ CS $]\!]$ $P_2 \sqsubseteq P_3$ $[\![$ CS $]\!]$ P_2, then (Monotonicity of Refinement 6) L ; P_1 $[\![$ CS $]\!]$ P_2 $\sqsubseteq$ L ; P_3 $[\![$ CS $]\!]$ P_2]

$c_1 \wedge c_2 \Rightarrow$ true

= (P $\Rightarrow$ true) = true

true

= true □

Sub-Rule 5.

***Interleave Compound Process Left**:*

$$\frac{(c_1 \models P_1) \rightsquigarrow (c_2 \models P_3) \qquad OP = |||}{(c_1 \models P_1\ OP\ P_2) \rightsquigarrow (c_2 \models P_3\ OP\ P_2)}$$

Proof

$$\frac{(c_1 \models P_1) \rightsquigarrow (c_2 \models P_3) \qquad OP = |||}{(c_1 \models P_1\ OP\ P_2) \rightsquigarrow (c_2 \models P_3\ OP\ P_2)}$$

= [Interleaving is equivalent to Parallelism with an empty channel set]

$$\frac{(c_1 \models P_1) \rightsquigarrow (c_2 \models P_3) \qquad OP = [\![\{\ \}]\!]}{(c_1 \models P_1\ OP\ P_2) \rightsquigarrow (c_2 \models P_3\ OP\ P_2)}$$

= Parallelism Compound Process Left [4]

true □

Attached Rule 36.

***Compound Process Right (for OP** $\in \{ \Box, \sqcap, [\![CS]\!], ||| \}$):*

$$\frac{(c_1 \models P_1) \rightsquigarrow (c_2 \models P_3) \qquad OP \in \{ \Box, \sqcap, [\![CS]\!], ||| \}}{(c_1 \models P_1\ OP\ P_2) \rightsquigarrow (c_2 \models P_1\ OP\ P_3)}$$

As, except for Sequence, all compound operators are commutative, the proof for attached rule 36 will be done re-using attached rule 35 as a theorem. The proof for Sequence

will be done on sub-rule 6.

Proof

$$\frac{(c_1 \models P_1) \rightsquigarrow (c_2 \models P_3) \qquad OP \in \{ \Box, \sqcap, [\![CS]\!], ||| \}}{(c_1 \models P_1 \ OP \ P_2) \rightsquigarrow (c_2 \models P_1 \ OP \ P_3)}$$

= [Except for Sequence, all compound operators are commutative ($\Box$, $\sqcap$, $[\![$ CS $]\!]$, $|||$)]

$$\frac{(c_1 \models P_1) \rightsquigarrow (c_2 \models P_3) \qquad OP \in \{ \Box, \sqcap, [\![CS]\!], ||| \}}{(c_1 \models P_2 \ OP \ P_1) \rightsquigarrow (c_2 \models P_3 \ OP \ P_1)}$$

= [Compound Process Left 35]

true □

Sub-Rule 6.

***Sequence Compound Process Right**:*

$$\frac{(c_1 \models P_1) \rightsquigarrow (c_2 \models P_3) \qquad OP = ;}{(c_1 \models P_1 \ OP \ P_2) \rightsquigarrow (c_2 \models P_1 \ OP \ P_3)}$$

In order to facilitate the proof, we will create and prove lemma B.10 and then prove sub-rule 6.

Proof

$(c_1 \models P_1 \ OP \ P_2) \rightsquigarrow (c_2 \models P_1 \ OP \ P_3)$

= [Lemma Compound Process B.10]

$c_1 \wedge c_2 \Rightarrow (\text{Lift } (gA \ (P_1)) \wedge \text{Lift } (gA \ (P_2)) \ ; P_1 \ OP \ P_2 \sqsubseteq (\text{Lift } (gA \ (P_1)) \wedge \text{Lift } (gA \ (P_2)) \ ; P_1 \ OP \ P_3))$

= [Be L = (Lift (gA (P_1)) $\wedge$ Lift (gA (P_2))]

$c_1 \wedge c_2 \Rightarrow L \ ; P_1 \ OP \ P_2 \sqsubseteq L \ ; P_1 \ OP \ P_3$

= [Assumption OP = ;]

$c_1 \wedge c_2 \Rightarrow L \ ; P_1 \ ; P_2 \sqsubseteq (L \ ; P_1 \ ; P_3)$

= [$P_1 \sqsubseteq P_3$ (Assumption 5) $\wedge$ $P_2 \sqsubseteq P_2$, so P_1 ; $P_2 \sqsubseteq P_1$; P_3 (6)]

$c_1 \wedge c_2 \Rightarrow L \ ; P_1 \ ; P_2 \sqsubseteq L \ ; P_1 \ ; P_3$ [having $P_1 \ ; P_2 \sqsubseteq P_1 \ ; P_3$]

= [L $\sqsubseteq$ L $\wedge$ P_1 ; $P_2 \sqsubseteq P_1$; P_3, then (Monotonicity of Refinement 6) L ; P_1 ; $P_2 \sqsubseteq$ L ; P_1 ; P_3]

$c_1 \wedge c_2 \Rightarrow$ true

= (P $\Rightarrow$ true) = true

true

= true □

B.3.18 Hide Process

Attached Rule 37.

Hiding Advance:

$$\frac{(c_1 \models P_1) \rightsquigarrow (c_2 \models P_2)}{(c_1 \models P_1 \setminus S) \rightsquigarrow (c_2 \models P_2 \setminus S)}$$

From Assumptions B.19:

[Assumption 1]: $(c_1 \models P_1) \rightsquigarrow (c_2 \models P_2)$
[Assumption 2]: Lift (getAssignments (P_1)) = Lift (getAssignments (P_2))
[Assumption 3]: Lift (getAssignments (P_1)) ; $P_1 \sqsubseteq$ Lift (getAssignments (P_2)) ; P_2)
[Assumption 4]: Lift (getAssignments (P_2)) ; P_1 Lift (getAssignments (P_2)) ; P_2
[Assumption 5]: $P_1 \sqsubseteq P_2$

Proof
$(c_1 \models P_1 \setminus S) \rightsquigarrow (c_2 \models P_2 \setminus S)$
= [Definition of Syntactic Transition B.27]
$\forall$ w . $c_1 \wedge c_2 \Rightarrow$
((Lift (getAssignments ($P_1 \setminus S$)) ; $P_1 \setminus S \sqsubseteq$ Lift (getAssignments ($P_2 \setminus S$)) ; $P_2 \setminus S$))
$\wedge$ Lift (getAssignments ($P_1 \setminus S$)) = Lift (getAssignments ($P_2 \setminus S$))
= [w is universally quantified, so we abstract it]
$c_1 \wedge c_2 \Rightarrow$
((Lift (getAssignments ($P_1 \setminus S$)) ; $P_1 \setminus S \sqsubseteq$ Lift (getAssignments ($P_2 \setminus S$)) ; $P_2 \setminus S$))
$\wedge$ Lift (getAssignments ($P_1 \setminus S$)) = Lift (getAssignments ($P_2 \setminus S$))
= [Definition of getAssignments (P $\setminus$ CS) B.1]
$c_1 \wedge c_2 \Rightarrow$
((Lift (getAssignments (P_1)) ; $P_1 \setminus S \sqsubseteq$ Lift (getAssignments (P_2)) ; $P_2 \setminus S$))
$\wedge$ Lift (getAssignments (P_1)) = Lift (getAssignments (P_2))
= [Assumption 2]
$c_1 \wedge c_2 \Rightarrow$
((Lift (getAssignments (P_1)) ; $P_1 \setminus S \sqsubseteq$ Lift (getAssignments (P_2)) ; $P_2 \setminus S$))
$\wedge$ true
= [P $\wedge$ true = P]
$c_1 \wedge c_2 \Rightarrow$
((Lift (getAssignments (P_1)) ; $P_1 \setminus S \sqsubseteq$ Lift (getAssignments (P_2)) ; $P_2 \setminus S$))
= [Assumption 2 again]

$c_1 \wedge c_2 \Rightarrow$
((Lift (getAssignments (P_1)) ; $P_1 \setminus S \sqsubseteq$ Lift (getAssignments (P_1)) ; $P_2 \setminus S$))
= [Assumption 5 and Hiding Monotonic B.2.9 $\Rightarrow$ $P_1 \setminus S \sqsubseteq P_2 \setminus S$. That with General Monotonicity of Refinement $\Rightarrow$ L ; P $\sqsubseteq$ L ; Q]
$c_1 \wedge c_2 \Rightarrow$ true
= [Predicate Calculus: P $\Rightarrow$ true = true]
$\forall$ w . true
=
true □

Attached Rule 38.

***Hiding Basic Process**:*
(c $\models$ *(***begin state** *ST PARS* • *A* **end***)* $\setminus$ *S)*
$\rightsquigarrow$
(c | *s* $\models$ **begin state** *ST PARS* • *(A* $\setminus$ *S)* **end***)*

Be HP = (**begin state** ST PARS • A **end**) $\setminus$ S

and

HB = **begin state** ST PARS • (A $\setminus$ S) **end**
Proof

$$\left(\begin{array}{l} (c \models (\textbf{begin state}\ ST\ PARS \bullet A\ \textbf{end}) \setminus S) \\ \rightsquigarrow \\ (c \mid s \models \textbf{begin state}\ ST\ PARS \bullet (A \setminus S)\ \textbf{end}) \end{array}\right)$$

= [Sigma Equal Basic Process 4 (having, from law B.2.19, that HP = HB]
true □

B.3.19 Rename Process

Attached Rule 39.

Rename Advance:

$$\frac{(c_1 \models P_1) \rightsquigarrow (c_2 \models P_2)}{(c_1 \models P_1\ [a_1, a_2, ... = b_1, b_2, ...]) \rightsquigarrow (c_2 \models P_2\ [a_1, a_2, ... = b_1, b_2, ...])}$$

From Assumptions B.19:

[Assumption 1]: $(c_1 \models P_1) \rightsquigarrow (c_2 \models P_2)$
[Assumption 2]: Lift (gA (P_1)) = Lift (gA (P_2))
[Assumption 3]: Lift (gA (P_1)) ; $P_1 \sqsubseteq$ Lift (gA (P_2)) ; P_2)
[Assumption 4]: Lift (gA (P_2)) ; P_1 Lift (gA (P_2)) ; P_2
[Assumption 5]: $P_1 \sqsubseteq P_2$

Proof

$(c_1 \models P_1\ [a_1, a_2, ... = b_1, b_2, ...]) \rightsquigarrow (c_2 \models P_2\ [a_1, a_2, ... = b_1, b_2, ...])$
= [Definition of Syntactic Transition B.27]

$$\forall w . c_1 \wedge c_2 \Rightarrow \left(\begin{array}{l} \left(\begin{array}{l} (Lift\,(gA\,(P_1\,[a_1, a_2, ... = b_1, b_2, ...]))\,;\ P_1\,[a_1, a_2, ... = b_1, b_2, ...]) \\ \sqsubseteq\ (Lift\,(gA\,(P_2\,[a_1, a_2, ... = b_1, b_2, ...]))\,;\ P_2\,[a_1, a_2, ... = b_1, b_2, ...]) \end{array} \right) \\ \wedge\ Lift\,(gA\,(P_1\,[a_1, a_2, ... = b_1, b_2, ...])) = Lift\,(gA\,(P_2\,[a_1, a_2, ... = b_1, b_2, ...])) \end{array} \right)$$

= [w is universally quantified, so we abstract it]

$$c_1 \wedge c_2 \Rightarrow \left(\begin{array}{l} \left(\begin{array}{l} (Lift\,(gA\,(P_1\,[a_1, a_2, ... = b_1, b_2, ...]))\,;\ P_1\,[a_1, a_2, ... = b_1, b_2, ...]) \\ \sqsubseteq\ (Lift\,(gA\,(P_2\,[a_1, a_2, ... = b_1, b_2, ...]))\,;\ P_2\,[a_1, a_2, ... = b_1, b_2, ...]) \end{array} \right) \\ \wedge\ Lift\,(gA\,(P_1\,[a_1, a_2, ... = b_1, b_2, ...])) = Lift\,(gA\,(P_2\,[a_1, a_2, ... = b_1, b_2, ...])) \end{array} \right)$$

= [Definition of gA (P [a_1, a_2, ... = b_1, b_2, ...]) B.1]

$$c_1 \wedge c_2 \Rightarrow \left(\begin{array}{l} \left(\begin{array}{l} (Lift\,(gA\,(P_1))\,;\ P_1\,[a_1, a_2, ... = b_1, b_2, ...]) \\ \sqsubseteq\ (Lift\,(gA\,(P_2))\,;\ P_2\,[a_1, a_2, ... = b_1, b_2, ...]) \end{array} \right) \\ \wedge\ Lift\,(gA\,(P_1)) = Lift\,(gA\,(P_2)) \end{array} \right)$$

= [Assumption 2]

$$c_1 \wedge c_2 \Rightarrow \left(\begin{array}{l} ((Lift\,(gA\,(P_1))\,;\ P_1\,[a_1, a_2, ... = b_1, b_2, ...]) \sqsubseteq (Lift\,(gA\,(P_2))\,;\ P_2\,[a_1, a_2, ... = b_1, b_2, ...])) \\ \wedge\ true \end{array} \right)$$

= [Predicate Calculus: P $\wedge$ true = P]

$c_1 \wedge c_2 \Rightarrow$
$((Lift\,(gA\,(P_1))\,;\ P_1\,[a_1, a_2, ... = b_1, b_2, ...] \sqsubseteq Lift\,(gA\,(P_2))\,;\ P_2\,[a_1, a_2, ... = b_1, b_2, ...]))$
= [Assumption 2 again]
$c_1 \wedge c_2 \Rightarrow$
$((Lift\,(gA\,(P_1))\,;\ P_1\,[a_1, a_2, ... = b_1, b_2, ...] \sqsubseteq Lift\,(gA\,(P_1))\,;\ P_2\,[a_1, a_2, ... = b_1, b_2, ...]))$
= [Assumption 5 and Rename Monotonic $\Rightarrow$ $\mathbf{P_1\ [a_1, a_2, ... = b_1, b_2, ...] \sqsubseteq P_2\ [a_1, a_2, ... = b_1, b_2, ...]}$. That with General Monotonicity of Refinement $\Rightarrow$ L ; P $\sqsubseteq$ L ; Q]
$c_1 \wedge c_2 \Rightarrow true$
= [Predicate Calculus: P $\Rightarrow$ true = true]
true □

Attached Rule 40.

Rename Basic Process:
(c $\models$ *(***begin state** *ST PARS* • *A* **end***) [a_1, a_2, ... = b_1, b_2, ...])*
$\rightsquigarrow$
(c | *s* $\models$ **begin state** *ST PARS* • *A [a_1, a_2, ... = b_1, b_2, ...]* **end***)*

Proof

Be

RP = (**begin state** ST PARS • A **end**) [a_1, a_2, ... = b_1, b_2, ...]

and

RB = **begin state** ST PARS • A [a_1, a_2, ... = b_1, b_2, ...] **end**

$$\left(\begin{array}{l} (c \models (\textbf{begin state}\ ST\ PARS \bullet A\ \textbf{end})\,[a_1, a_2, ... = b_1, b_2, ...]) \\ \rightsquigarrow \\ (c \mid s \models \textbf{begin state}\ ST\ PARS \bullet (A\,[a_1, a_2, ... = b_1, b_2, ...])\ \textbf{end}) \end{array}\right)$$

= [Sigma Equal Basic Process 4 (having, from law B.2.18, that RP = RB]

true □

B.3.20 Call Process

Attached Rule 41.

Parameterless Call Process:
(c ⊨ P) ⇝ (c ⊨ Content (P))

Proof (c ⊨ P) ⇝ (c ⊨ Content (P))
= [Definition B.8]
(c ⊨ Content (P)) ⇝ (c ⊨ Content (P))
= [Sigma Equal Sides 3]
true □

Attached Rule 42.

Call Process with normal parameters:
(c | s ⊨ P (N+)) ⇝ (c ⊨ ParamContent (P, [N+]))

Proof
(c ⊨ P) ⇝ (c ⊨ Content (P, [N+]))
= [Definition B.9]
(c ⊨ ParamContent (P, [N+])) ⇝ (c ⊨ ParamContent (P, [N+]))
true □

Attached Rule 43.

Call Process with indexed parameters:
(c ⊨ P ⌊ N+ ⌋) ⇝ (c ⊨ IndexedContent (P, [N+]))

Proof
(c ⊨ P ⌊ N+ ⌋) ⇝ (c ⊨ IndexedContent (P, [N+]))
= [Definition B.10]
(c ⊨ IndexedContent (P, [N+])) ⇝ (c ⊨ IndexedContent (P, [N+]))
= [Sigma Equal Sides 3]
true □

Attached Rule 44.

Call Process with generic parameters:
(c ⊨ P [N+]) ⇝ (c ⊨ GenericContent (P, [N+]))

Proof
(c ⊨ P [N+]) ⇝ (c ⊨ GenericContent (P, [N+]))

= [Definition B.11]
(c $\models$ GenericContent (P, [N+])) $\rightsquigarrow$ (c $\models$ GenericContent (P, [N+]))
= [Sigma Equal Sides 3]
true □

B.3.21 Iterated Process

Attached Rule 45.

***Iterated Processes**:*
(c $\models$ OP Decl • P) $\rightsquigarrow$ (c $\models$ IteratedExpansion (P, Decl, ITOPFLAG))
IteratedExpansion is defined on B.3.

Proof (c $\models$ OP Decl • P) $\rightsquigarrow$ (c $\models$ IteratedExpansion (P, Decl, ITOPFLAG))
= [Definition of Iterated Operator B.3]
(c $\models$ IteratedExpansion (P, Decl, ITOPFLAG)) $\rightsquigarrow$ (c $\models$ IteratedExpansion (P, Decl, ITOPFLAG))
= [Silent Transition Equal Sides 3]
true □

Bibliography

[1] Ana Cavalcanti and Marie-Claude Gaudel. Testing for refinement in Circus. *Acta Inf.*, 48(2):97–147, 2011.

[2] Ana Cavalcanti and Jim Woodcock. A tutorial introduction to csp in unifying theories of programming. *Refinement techniques in software engineering*, pages 220–268, 2006.

[3] Ana Cavalcanti and Jim Woodcock. CML Definition 2 - Operational Semantics (Deliverable Number: D23.3-4, Version 0.2). Technical report, University of York, 2013.

[4] C. A. R. Hoare and He Jifeng. *Unifying Theories of Programming*. Prentice-Hall, 1998.

[5] M Oliveira, A Sampaio, PRG Antonino, RT Ramos, A Cavalcanti, and J Woodcock. Compositional analysis and design of cml models. *COMPASS Deliverable D*, 24, 2013.

[6] M. V. M. Oliveira. *Formal Derivation of State-Rich Reactive Programs using Circus*. PhD thesis, Department of Computer Science, University of York, 2006. Oliveira.

[7] M. V. M. Oliveira. *Formal Derivation of State-Rich Reactive Programs using Circus - Extended Version*. PhD thesis, Department of Computer Science, University of York, 2006. Oliveira.

[8] J. C. P. Woodcock and A. L. C. Cavalcanti. A Tutorial Introduction to Designs in Unifying Theories of Programming. In E. A. Boiten, J. Derrick, and G. Smith, editors, *IFM 2004: Integrated Formal Methods*, volume **2999** of *Lecture Notes in Computer Science*, pages 40–66. Springer-Verlag, 2004. Invited tutorial.

[9] J. C. P. Woodcock and J. Davies. *Using Z—Specification, Refinement, and Proof*. Prentice-Hall, 1996.