New *Circus Time*

Kun Wei

Department of Computer Science
University of York

Technical Report

April 2013

Abstract

The *Circus* family of languages is a collection of process algebras based on a combination of Z, for data modelling, and CSP, for behavioural modelling. *Circus* has a well-defined semantics based on Hoare and He's Unifying Theories of Programming. A refinement calculus has enabled its practical use for reasoning about implementations of control systems in a variety of languages, including Ada and Java. The work presented in this report enables the use of pre and postconditions for reasoning in the context of *Circus Time*, the timed version of *Circus*. It is based on constructs of Timed CSP, but takes advantage of the complete lattice defined by its UTP theory to cater for deadlines that capture restrictions on the environment. Here, we present a new UTP theory that builds on the existing semantics of *Circus Time* by providing a simpler mathematical model for capturing observations of traces and refusals, and more rigorous operators for describing behaviours of processes. For example, we change the definitions of prefix and external choice by weakening their preconditions and strengthening postconditions. Most importantly, all operators of our theory are defined in a way that allows the calculation of pre and postconditions of timed processes. This not only supports contract-based reasoning about models, but simplifies proof of *Circus Time* laws. We illustrate this point by exploring a comprehensive set of laws for the deadline operators.

Contents

List of Figures

List of Tables

1 Introduction

Circus [27, 4] is a comprehensive combination of Z [25], CSP [9, 17] and Morgan's refinement calculus [12], so that it can define both data and behavioural aspects of a system. Over the years, *Circus* has developed into a family of languages for specification, programming and verification. For instance, variants and extensions of *Circus* include *Circus Time* [20, 21], which provides time facilities similar to Timed CSP [18], and *OhCircus* [5], which is based on the Java approach to object-orientation. The semantics of the *Circus* languages is based on Hoare and He's Unifying Theories of Programming (UTP) [10], in which they use the alphabetised relational calculus to describe and reason about a wide variety of programming paradigms.

Circus Time [20, 21] is an extension of a subset of *Circus* with some time operators. The semantics of *Circus Time* is defined using the UTP by introducing time observation variables. The *Circus Time* UTP theory is a discrete time model, and time operators are very similar to those in Timed CSP [18]. Simply speaking, the observation of an action in *Circus Time* is distributed into a sequence of time units. Multiple events are allowed to happen within the same time unit, but they must be ordered. The major observation over a time unit consists of a sequence of events that have happened during the time unit, and a refusal set that denotes which events cannot happen at the end of this time unit. One of features of *Circus Time* is that it can be seamlessly combined with Z specifications in *Circus* to provide better support for modelling and refinement of real-time systems. In addition, *Circus Time* also provides a number of refinement laws and a framework for validation of timed programs based on FDR [1].

Circus Time cannot, however, capture some key requirements in safety-critical systems such as missing of a deadline, can result in significant loss to its users. Here, a (hard) deadline imposes requirements on a system's environment, such as hardware and changes of physical environment, whose knowledge is hard to be aware by a designer in an abstraction level. Therefore, in this paper we extend the original *Circus Time* model with more time operators. For example, two deadline operators have been introduced in this new model: one is that an action must terminate within a deadline, the other is that a visible interaction of the action must occur within the deadline. Similar operators have been defined in existing process algebras and temporal logic [7, 19, 16], but none of them have explored any possible influence to other existing operators. We adopt a similar idea to that in [7]; that is, failure to meet a deadline leads to infeasibility in the model of a program. The infeasibility in our new *Circus Time* is modelled by a new primitive action, *Miracle*. This action, the top element in the refinement ordering, has been included in the semantics of the original *Circus Time* model, but its use to model deadlines was only exposed and initially explored in the work [24] who discussed the application of *Miracle* in a complete lattice and used it to express the infeasibility when deadlines cannot be satisfied. In this paper, inspired by the work in in [24], we fully explore this action and also its interaction the other operators by posing and proving a number of algebraic laws to characterise a theory of deadlines.

Another difference from the original *Circus Time* is that each action in our new *Circus Time* is expressed as a reactive design for a more concise, readable and uniform UTP semantics. Hoare and He have proposed an approach to generate the theory of CSP by embedding the theory of designs in the theory of reactive processes. This means that each process in CSP can be expressed as a reactive design. The work in [3, 13] has provided reactive design semantics for some operators in CSP. On the other hand, sequential composition, recursion, hiding, and so on have not been considered. In this paper, in our new *Circus Time* model, we develop the reactive design semantics of these operators, and also demonstrate how we can easily prove some very tricky laws using the new semantics. The importance of this semantics is that it not only supports contract-based reasoning about models, but simplifies proof of *Circus Time* laws. For instance, reactive designs can be easily related to Hoare logic [8], which is a formal system for reasoning about the correctness of computer programs. As a result, we present a new UTP theory that builds on the existing semantics of *Circus Time* by providing a simpler mathematical model for capturing observations of traces and refusals, and more rigorous operators for describing behaviours of processes. For instance, we change the definitions of prefix and external choice by weakening their preconditions and strengthening postconditions.

rewrite the following paragraph.

This report has the following structure. Section 2 gives a detailed introduction to various UTP theories which have been used in the new model. Also, the semantics of the old version of *Circus Time* is discussed and modified. We then present a new model of *Circus Time* via changing the data structure of some observational variables, redefining healthiness conditions and introducing new time operators in Section 3. Section 4 explores the relation between the old and new versions of *Circus Time*. Finally in Section 5 we present some conclusions and summary future work.

2 Background

This section introduces various UTP theories such as relations, designs, reactive processes, CSP processes and reactive designs which are used throughout the semantics of *Circus Time* in this report. In addition, before extending the original *Circus Time*, we introduce its semantics including observational variables, healthiness conditions and signature.

2.1 Relations

In the UTP, a relation P is a predicate with an alphabet αP, composed of *undashed* variables $(a, b, ...)$ and *dashed* variables $(a', x', ...)$. The former, written as $in\alpha P$, stands for initial observations, and the latter, $out\alpha P$, for intermediate or final observations. The relation is called *homogeneous* if $out\alpha P = in\alpha P'$, where $in\alpha P'$ is obtained by dash on all the variables of $in\alpha P$. A *condition* has an empty output alphabet.

The program constructors in the theory of relations include sequential composition $(P\,;Q)$, conditional $(P \lhd b \rhd Q)$, assignment $(x := e)$, nondeterminism $(P \sqcap Q)$ and recursion $(\mu\, X \bullet C(X))$. The correctness of a program P with respect to a specification S is denoted by $S \sqsubseteq P$ (P refines S), and is defined as $S \sqsubseteq P$ if, and only if, $[P \Rightarrow S]$. Here, the square bracket is universal quantification over all variables in the alphabet. In other words, the correctness of P is proved by establishing that every observation that satisfies P must also satisfy S. Moreover, the set of relations with a particular alphabet is a complete lattice under the refinement ordering. Its bottom element is the weakest relation **true**, which behaves arbitrarily ($\mathbf{true} \sqsubseteq P$), and the top element is the strongest relation **false**, which behaves miraculously and satisfies any specification ($P \sqsubseteq \mathbf{false}$).

The least upper bound is not defined in terms of the relational model, but by Law 1, the law that we reproduce below as and which applies equally to infinite and to finite nondeterministic choice . This law is introduced by an axiom rather than a formal definition in the UTP. Law 1 may be easier to understand when it is split into two sub-laws, L1A and L1B.

Law 1. *Let S be some sets of predicates,*

$$P \sqsubseteq (\bigsqcap S) \textbf{ iff } (P \sqsubseteq X \text{ for all } X \text{ in } S)$$

L1A $(\bigsqcap S) \sqsubseteq X$ for all X in S

L1B if $P \sqsubseteq X$ for all X in S, then $P \sqsubseteq (\bigsqcap S)$

These laws state that the disjunction of a set is a lower bound of all its members by L1A, and it is also the greatest lower bound (*glb*) by L1B. The bottom and top elements in this complete lattice are usually called *Chaos* and *Miracle* respectively. For a tutorial introduction to relations, we refer to [10, 26].

2.2 Designs

A design in the UTP is a relation that can be expressed as a pre-postcondition pair in combination with boolean variables, called ok and ok'. In designs, ok records that the program has started, and ok' records that it has terminated. If a precondition P and a postcondition Q are predicates not containing ok and ok', a design with P and Q, written as $P \vdash Q$, is defined as follows:

$$P \vdash Q \mathrel{\widehat{=}} ok \wedge P \Rightarrow ok' \wedge Q$$

which means that if a program starts in a state satisfying P, then it must terminate, and whenever it terminates, it must satisfy Q.

Healthiness conditions of a theory in the UTP are a collection of some fundamental laws that must be satisfied by relations belonging to the theory. These laws are expressed in terms of monotonic idempotent functions. The healthy relations are the fixed points of these functions. There are four healthiness conditions identified by Hoare and He in the theory of designs.

$$\mathbf{H1}(P) = ok \Rightarrow P \qquad \mathbf{H2}(P) = [P[\mathbf{false}/ok'] \Rightarrow P[\mathbf{true}/ok']]$$
$$\mathbf{H3}(P) = P\,;\, \mathbb{I}_D \qquad \mathbf{H4}(P) = [P;\, \mathbf{true}]$$

The first healthiness condition means that observations of a predicate P can only be made after the program has started. **H2** states that a design cannot require non-termination, since if P is satisfied when ok' is false,

it must also be satisfied when ok' is true. Moreover, **H2** can also be expressed by removing the universal quantifier, $P = P\,;\, J$, in which $J \mathrel{\widehat{=}} (ok \Rightarrow ok') \wedge v' = v$, and v and v' in the alphabet of J denotes all the variables except for ok and ok'. A predicate is **H1** and **H2** if, and only if, it is a design; the proof is in [10].

A design is **H3** only when its precondition is a condition. The design identity is defined as $\mathit{II}_D \mathrel{\widehat{=}} \mathbf{true} \vdash \mathit{II}$ where II is the relational identity. The relational identity always terminates and leaves the values of all the variables unchanged, defined as $\mathit{II} \mathrel{\widehat{=}} v' = v$ where $\alpha\mathit{II} = \{v, v'\}$. The final healthiness condition **H4** is actually equivalent to $[\exists\, ok', v' \bullet P]$ which means that for every initial value of the observational variables in the alphabet, there exist final values of the variables. In other words, a program P satisfies **H4** only if establishing a final state is feasible. For example, the design miracle ($\mathbf{true} \vdash \mathbf{false}$) does not satisfy **H4**, and the proof is given as follows

Proof.

$$
\begin{array}{ll}
\quad \mathbf{H4}(\mathbf{true} \vdash \mathbf{false}) & [\mathbf{H4}] \\
= [\exists\, ok', v' \bullet \mathbf{true} \vdash \mathbf{false}] & [\text{Design}] \\
= [\exists\, ok', v' \bullet ok \wedge \mathbf{true} \Rightarrow ok' \wedge \mathbf{false}] & [\text{propositional calculus}] \\
= [\exists\, ok', v' \bullet \neg\, ok] & [\text{predicate calculus}] \\
= [\neg\, ok] & [\text{case split}] \\
= \neg\, \mathbf{true} \wedge \neg\, \mathbf{false} & [\text{propositional calculus}] \\
= \mathbf{false} &
\end{array}
$$

□

If any of the program operators are applied to designs, then the combination is also a design, which follows the laws from the UTP book [10], for choice, conditional, sequential composition and recursion.

Property 1. *(Designs)*

L1. $((P_1 \vdash P_2) \sqcap (Q_1 \vdash Q_2)) = (P_1 \wedge Q_1 \vdash P_2 \vee Q_2)$

L2. $((P_1 \vdash P_2) \lhd b \rhd (Q_1 \vdash Q_2)) = ((P_1 \lhd b \rhd Q_1) \vdash (P_2 \lhd b \rhd Q_2))$

L3. $((P_1 \vdash P_2)\,;\, (Q_1 \vdash Q_2)) = ((\neg\, (\neg\, P_1\,;\, \mathit{true}) \wedge \neg\, (P_2\,;\, \neg\, Q_1)) \vdash (P_2\,;\, Q_2))$

L4. $(\mu\, X, Y \bullet (F(X, Y) \vdash G(X, Y))) = (P(Q) \vdash Q)$

where $P(Y) = \nu X \bullet F(X, Y)$ ***and*** $Q = \mu\, Y \bullet P(Y) \Rightarrow G(P(Y), Y)$

The purpose of the theory of designs is to exclude relations that satisfy the equation $\mathbf{true}\,;\, P = P$. The program **true** models abort. For example, the least fixed-point semantics of a nonterminating loop in the theory of relations is **true**, and, when followed by an assignment like $x := c$, behaves like the assignment. In practice, it means that a program could recover from a nonterminating loop. Therefore, the theory of designs is developed to make any element of the theory satisfy the expected equation, $\mathbf{true}\,;\, P = \mathbf{true}$. For a tutorial introduction to designs, the reader is referred to [10, 26].

2.3 Reactive processes

A reactive process is a program whose behaviour may depend on interactions with its environment. To represent intermediate waiting states, a boolean variable *wait* is introduced to the alphabet of a reactive process. For example, if $wait'$ is true, then the process is in an intermediate state. If *wait* is true, we have an intermediate observation of its predecessor. Thus, we are able to represent any states of a process by combining the values of ok and *wait*. If ok' is false, the process diverges. If ok' is true, the state of the process depends on the value of $wait'$. If $wait'$ is true, the process is in an intermediate state; otherwise it has successfully terminated. Similarly, the values of undashed variables represent the states of a process's predecessor.

Apart from ok, ok', *wait* and $wait'$, another two pairs of observational variables, *tr* and *ref*, and their dashed counterparts, are introduced. The variable *tr* records the events that have occurred until the last observation, and tr' contains all the events since that observation. Similarly, *ref* records the set of events that could be refused in the last observation, and ref' records the set of events that may be refused currently. The reactive identity, II_{rea}, is defined as follows:

$$\mathit{II}_{rea} \mathrel{\widehat{=}} (\neg\, ok \wedge tr \leq tr') \vee (ok' \wedge wait' = wait \wedge tr' = tr \wedge ref' = ref)$$

A reactive process must satisfy the following healthiness conditions:

$$\mathbf{R1}(P) = P \wedge tr \leq tr' \qquad \mathbf{R2}(P(tr, tr')) = \bigsqcap s \bullet P(s, s \frown (tr' - tr)) \qquad \mathbf{R3}(P) = \mathbb{I}_{rea} \lhd wait \rhd P$$

If a relation P describes a reactive process behaviour, **R1** states that it never changes history, or the trace is always increasing. The second, **R2**, states that the history of the trace tr has no influence on the behaviour of the process. As the work in [3], **R2** can also be expressed as $P(tr, tr') = P(\langle\rangle, tr' - tr)$, which has the same set of fixed points as that of the original one. The final, **R3**, requires that a process should leave the state unchanged ($\mathbb{I}_{rea}$) if it is in a waiting state ($wait = true$) of its predecessor. A reactive process is a relation whose alphabet includes ok, $wait$, tr and ref, and their dashed counterparts, and is a fixed point of the composition $\mathbf{R}$ where $\mathbf{R} \mathrel{\widehat{=}} \mathbf{R1} \circ \mathbf{R2} \circ \mathbf{R3}$. Since each of $\mathbf{Ri}$ is idempotent and any two of them commute, $\mathbf{R}$ is also idempotent. For a more detailed introduction to the theory of reactive designs, the reader is referred to the tutorial [3].

2.4 CSP processes

In the UTP, the theory of CSP is built by applying extra healthiness conditions to reactive processes. For example, a reactive process is also a CSP process if and only if, it satisfies the following healthiness conditions:

$$\mathbf{CSP1}(P) = P \vee (\neg\, ok \wedge tr \leq tr') \qquad \mathbf{CSP2}(P) = P\,;\, J$$

where J has been given in Section 2.2 and here is given by unfolding $v' = v$ as $wait' = wait \wedge tr' = tr \wedge ref' = ref$. The first healthiness condition requires that, in case of divergence of the predecessor, the extension of the trace should be the only guaranteed property. The second one means that P cannot require non-termination, so that it is always possible to terminate.

The CSP theory introduced in the UTP book is different from any standard models of CSP [9, 17] which have more restrictions or satisfy more healthiness conditions. There are more healthiness conditions, **CSP3**-**CSP5**, given in the UTP to further restrict behaviours of reactive processes.

$$\mathbf{CSP3}(P) = Skip\,;\, P \qquad \mathbf{CSP4}(P) = P\,;\, Skip \qquad \mathbf{CSP5}(P) = P \,|||\, Skip$$

The process $Skip$, defined in the UTP as $Skip \mathrel{\widehat{=}} \mathbf{R}(\exists\, ref \bullet \mathbb{I})$, is a primitive process that refuses to engage in any communication event, but terminates immediately. Thus, **CSP3** requires that the initial value of ref has no influence on the behaviour of a process, and it in fact satisfies $\neg\, wait \Rightarrow (P = \exists\, ref \bullet P)$. It is proved in the UTP book that P is **CSP3**, if and only if, it satisfies the left unit law, $Skip\,;\, P = P$. Similarly, **CSP4** states that the value of ref' is irrelevant on termination or divergence.

That $|||$ in CSP is a parallel operator that combines interleaving concurrently without any interaction. As a matter of fact, $P \,|||\, Q$ can refuse an event only when it is refused by both processes. For example, if we use $0.ref$ and $1.ref$ to denote the refusals of P and Q respectively, the refusal (ref') in the interleave is $0.ref \cap 1.ref$. As we know, ref' in $Skip$ is arbitrary and consequently **CSP5** states that if a process can refuse a set X of events, then it can also refuse any subset of X. In addition, it has been proved that all CSP operators given in the UTP book [10] satisfy **CSP3**-**CSP5**.

A CSP process can also be obtained by applying the healthiness condition $\mathbf{R}$ to a design. This follows from the theorem in [10] that establishes that for every CSP process P, $P = \mathbf{R}(\neg\, P^f_f \vdash P^t_f)$. This theorem gives a new style of specification for CSP processes in which a design describes the behaviour when its predecessor has terminated and not diverged, and the other situations of its behaviour are left to $\mathbf{R}$. We use P^a_b as an abbreviation of $P[a, b/ok', wait]$. Motivated by the above theorem, the work in [3, 13] provide reactive design definitions for some constructs of CSP. Besides redefining these operators in a timed environment, the reactive design definitions of more CSP operators, such as sequential composition, hiding and recursion, are developed in this paper.

2.5 *Circus*

Circus programs are formed by a sequence of paragraphs: channel declarations, channel set definition, Z paragraphs, or process definitions, in which processes are the key elements of *Circus* specifications. Unlike CSP, a *Circus* process consists of behaviours, constructed by CSP operators as actions, and local states, defined by Z schemas and accessible only by its local actions, but hidden to other processes. Processes can

communicate each other through channels. The least elements in *Circus* specification are actions that can be Z schemas, guarded commands of Dijkstra's language, and CSP processes. In brief, a *Circus* program can be roughly considered a mixture of Z schemas, imperative commands and CSP processes.

The semantics of *Circus* is based on the UTP, firstly proposed in [28] and later enhanced in [13] for better mechanisation. In the UTP, designs, specifications and programs are all interpreted as relations (or alphabetised predicates) with an initial observation and a subsequent (intermediate or final) observation. That is very similar to the decoration style of Z schemas. All in all, the basic idea to integrate Z and CSP in *Circus* programs is to convert Z expressions into specification statements by the refinement calculus in [2] and then embed the date operation into the theory of reactive processes by the use of $\mathbf{R}$. In other words, we treat a *Circus* program as a predicate.

2.6 Original *Circus Time* Action Model

We give an introduction to *Circus Time* because the latter sections in this report often directly use definitions given in this section.

2.6.1 Alphabet

The *Circus Time* action model [20, 21] uses a subset of *Circus* and adds time operators to the notion of actions in *Circus*. The semantics of *Circus Time* is also defined using UTP by introducing time observation variables. *Circus Time* is a discrete time model in which observations are filled into a sequence of time units. Multiple events are allowed to happen within a same time unit, but they must be ordered. *Circus Time* has similar observational variables of *Circus*, but changes the structure of traces to capture more information.

In *Circus Time*, an action is described as an alphabetised predicate whose observational variables are as follows:

- $ok, ok' : Boolean$
- $wait, wait' : Boolean$
- $state, state' : N \to Value$
- $tr_t, tr'_t : seq^+(seq\ Event \times \mathbb{P}\ Event)$

where N is a set of names of variables, and timed traces tr_t and tr'_t are defined to be non-empty (seq^+), in which each element represents an observation over one time unit. Each observation is a pair, where the first element is a sequence of events that have occurred within the time unit, and the second is a refusal at the end of the same time unit. Thus, a time is actually hidden in the length of a timed trace. Note that in the original *Circus Time* the indices of a timed trace start with 0. In addition, another variable $trace'$ is used to record only a sequence of events that have occurred since the last observation.

$$\begin{aligned} &trace' : seq\ Event \\ &trace' = Flat(tr'_t) - Flat(tr_t) \end{aligned} \tag{2.1}$$

where *Flat* maps timed traces to untimed traces, as similar as $\frown/$ (distributed concatenation) on sequences. Here we use the mathematical notation of Z as a meta-language to describe our theory. We explain the details of the notation at the points where they are firstly used. For sequences, we use *head*, *tail*, *front*, *last*, #(length), $\frown$(concatenation) and $\frown/$(flattening). Another two extra operations, $\leq$ (prefix) and $-$ (difference), are defined as

$$s \leq t \Leftrightarrow \exists\, u \bullet t = s \frown u$$

$$t - s = u \Leftrightarrow t = s \frown u$$

Additionally, an expanding relation between traces in *Circus Time* is defined as follows

$$Expands(tr_t, tr'_t) \mathrel{\widehat=} (front(tr_t) \leq tr'_t) \land (fst(last(tr_t)) \leq fst(tr'_t(\#tr_t))) \tag{2.2}$$

2.6.2 Healthiness Conditions

Circus Time adopts all healthiness conditions of *Circus*, but of course adapts them for the new structure of timed traces. In addition, all functions on sequences can be found in the appendix or in the original work [21].

The healthiness condition $\mathbf{R1_t}$ states that a program X can never undo any actions that has been executed previously.

$$\mathbf{R1_t}(X) = X \land Expands(tr_t, tr'_t) \tag{2.3}$$

Here *Circus Time* uses the relation *Expands* rather than the prefix relation ($\leq$) because, different from traces whose observation can be clearly recognised in an action, the initial observation of refusal sets usually overrides the last observation of its predecessor. That is, the prefix relation between $snd(last(tr_t))$ and $snd(tr'_t(\#tr_t))$ is unnecessary.

The second condition $\mathbf{R2_t}$ states that tr_t has no influence on the behaviour of the program.

$$\mathbf{R2_t}(X) = \exists\, ref \bullet X[\langle(\langle\rangle, ref)\rangle, dif(tr'_t, tr_t)/tr_t, tr'_t] \tag{2.4}$$

$$dif(tr'_t, tr_t) \mathrel{\widehat{=}} \langle(fst(head(tr'_t - front(tr_t))) - fst(last(tr_t)), \tag{2.5}$$
$$snd(head(tr'_t - front(tr_t))))\rangle \frown tail(tr'_t - front(tr_t))$$

where *dif* is used to obtain the difference between two timed traces. Since a timed trace in *Circus Time* is a pair of a trace and a refusal, the empty trace ($\langle\rangle$) and the arbitrary *ref* states that the history of the trace and the refusal is irrelevant for the behaviour of the program.

The condition $\mathbf{R3_t}$ assures that if the program X is requested to start in a waiting state of its predecessor, it keeps the state unchanged.

$$\mathbf{R3_t}(X) = \mathit{I\!I}_t \lhd wait \rhd X \tag{2.6}$$

where the identity is defined as

$$\mathit{I\!I}_t = (\neg\, ok \land Expands(tr_t, tr'_t)) \lor (ok' \land tr'_t = tr_t \land wait' = wait \land state' = state) \tag{2.7}$$

Since a *Circus Time* program is also a CSP process, a timed reactive program must satisfy another five healthiness conditions.

$$\mathbf{CSP1_t}(X) = X \lor (\neg\, ok \land Expands(tr_t, tr'_t)) \tag{2.8}$$
$$\mathbf{CSP2_t}(X) = X \,;\, J_t \tag{2.9}$$
$$\mathbf{CSP3_t}(X) = Skip \,;\, X \tag{2.10}$$
$$\mathbf{CSP4_t}(X) = X \,;\, Skip \tag{2.11}$$
$$\mathbf{CSP5_t}(X) = X \;|||\; Skip \tag{2.12}$$
$$J_t = (ok \Rightarrow ok') \land wait' = wait \land tr'_t = tr_t \land state' = state \tag{2.13}$$

These healthiness conditions $\mathbf{R_t}(\mathbf{R_t} = \mathbf{R1_t} \circ \mathbf{R2_t} \circ \mathbf{R3_t})$ and $\mathbf{CSP1_t}$-$\mathbf{CSP5_t}$ have the same meaning as those in the CSP theory.

2.6.3 Syntax and Semantics of *Circus Time*

Figure 1 presents the description of the syntax of *Circus Time*. In this figure, b stands for a condition, c is a channel, e for any expression, d for a non-negative integer number, N for any valid name (or identifier), s for a set of variable names, and CS for a set of channel names.

$$\begin{aligned} P ::= & Skip \mid Stop \mid Chaos \mid c.e \rightarrow P \mid P \,;\, Q \mid P \lhd b \rhd Q \mid N := e \mid b \& P \mid P \,\Box\, Q \mid P \sqcap Q \\ & \mid P \,|[\, s_1 \mid \{\!|\, CS \,|\!\} \mid s_2 \,]|\, Q \mid P \setminus CS \mid Wait\ d \mid P \rhd\!\{d\}\ Q \mid \mu\, N \bullet P \end{aligned}$$

Figure 1: Original *Circus Time* Syntax

Skip, Stop and Chaos

The action *Stop* is a deadlocked process which is incapable of communicating with its environment, but allows time to pass and local variables to change.

$$Stop \mathrel{\widehat{=}} \mathbf{CSP1_t}(\mathbf{R3_t}(ok' \wedge wait' \wedge trace' = \langle\rangle)) \tag{2.14}$$

The action *Skip* terminates immediately without changing anything, and its definition in *Circus Time* is given as follows:

$$Skip \mathrel{\widehat{=}} \mathbf{R_t}(\exists\, ref \bullet ref = snd(last(tr_t)) \wedge \mathit{II}_t) \tag{2.15}$$

The action *Chaos* is the worst action whose behaviour can be anything but satisfies $\mathbf{R_t}$.

$$Chaos \mathrel{\widehat{=}} \mathbf{R_t}(\mathbf{true}) \tag{2.16}$$

Wait

The delay action *Wait* does nothing except that it allows an interval of time to pass. The definition of *Wait* in *Circus Time* is as follow:

$$Wait\ d \mathrel{\widehat{=}} \mathbf{CSP1_t}(\mathbf{R_t}(ok' \wedge delay(d) \wedge trace' = \langle\rangle)) \tag{2.17}$$

$$delay(d) \mathrel{\widehat{=}} \big(\ (wait' \wedge \#tr_t' - \#tr_t < d) \vee (\neg\, wait' \wedge \#tr_t' - \#tr_t = d \wedge state' = state)\ \big) \tag{2.18}$$

In addition, we give two lemmas that state that $delay(d)$ and $trace' = \langle\rangle$ are $\mathbf{R2_t}$ healthy. Because it is very easy to prove these lemmas, their proofs are not given here.

Lemma 1. $\mathbf{R2_t}(delay(d)) = delay(d)$

Lemma 2. $\mathbf{R2_t}(trace' = \langle\rangle) = (trace' = \langle\rangle)$

Sequential Composition

The sequential composition $P\,;\,Q$ behaves as P until P terminates, and then behaves as Q. In the meanwhile the final state of P is passed on as the initial state of Q. Its semantics is given as follows:

$$P\,;\,Q \mathrel{\widehat{=}} \exists\, obs_0 \bullet P[obs_0/obs'] \wedge Q[obs_0/obs] \quad \textit{if } out\alpha(P) = in\alpha(Q)' = \{obs'\} \tag{2.19}$$

where *obs* is a collection of all observational variables in this model.

Conditional Choice

The definition of the choice operator in *Circus Time* is the same as that in the UTP, which is defined as follows:

$$P \lhd b \rhd Q \mathrel{\widehat{=}} (b \wedge P) \vee (\neg\, b \wedge Q) \tag{2.20}$$

where b is a condition or does not contain any dashed variable.

Assignment

Suppose that x is a program variable that is included in v, and e is an expression. The notation $(x := e)$ represents the action that simply assigns the value of e to x and terminates immediately. Meanwhile, other variables within *state* keep unchanged.

$$x := e \mathrel{\widehat{=}} \mathbf{CSP1_t}(\mathbf{R_t}(ok' = ok \wedge wait' = wait \wedge tr_t' = tr_t \wedge state' = state \oplus \{x \mapsto val(e, state)\})) \tag{2.21}$$

Here, the evaluation function $val(e, state)$ returns the value of the expression e in *state*, and $\oplus$ is an overriding operator.

Guarded Action

In a guarded action $b\&P$, it will behave P if the condition b is true. Otherwise, it just behaves like *Stop*.

$$b\&P \mathrel{\widehat{=}} P \lhd b \rhd Stop \tag{2.22}$$

Communication

The prefix action $c.e \rightarrow P$ is able to pass the value e through the channel c and then behaves as P. This action can also be represented by a composition of a simple prefix and P itself, written as $(c.e \rightarrow Skip)\,;\,P$. Here, the semantics of the simple prefix in the *Circus Time* model is given as follows:

$$c.e \rightarrow Skip \mathrel{\widehat{=}} \mathbf{CSP1_t}(ok' \wedge \mathbf{R_t}(wait\_com(c) \vee terminating\_com(c.e))) \tag{2.23}$$
$$wait\_com(c) \mathrel{\widehat{=}} wait' \wedge possible(tr_t, tr'_t, c) \wedge trace' = \langle\rangle \tag{2.24}$$
$$possible(tr_t, tr'_t, c) \mathrel{\widehat{=}} \forall\, i : \#tr_t..\#tr'_t \bullet c \notin snd(tr'_t(i)) \tag{2.25}$$
$$term\_com(c.e) \mathrel{\widehat{=}} \neg\, wait' \wedge trace' = \langle c.e\rangle \wedge \#tr'_t = \#tr_t \tag{2.26}$$
$$terminating\_com(c.e) \mathrel{\widehat{=}} (wait\_com(c)\,;\, term\_com(c.e)) \vee term\_com(c.e) \tag{2.27}$$

The above definition can also be concluded that such an action may execute in three possible behaviours: the action is waiting to communicate on the channel c, or it is waiting in the beginning and then is followed by termination with the occurrence of $c.e$, or it simply terminates immediately after the start. Here, we give some lemmas that actually say $wait\_com(c)$ and $ter\_com(c.e)$ are $\mathbf{R2_t}$ healthy. These lemmas will be used for the transformation in Section 3.4, and their proofs will not be given here.

Lemma 3. $\mathbf{R2_t}(wait\_com(c)) = wait\_com(c)$

Lemma 4. $\mathbf{R2_t}(term\_com(c.e)) = term\_com(c.e)$

Lemma 5. $\mathbf{R2_t}(terminating\_com(c.e)) = terminating\_com(c.e)$

External Choice

The action $P \,\Box\, Q$ may behave either like the conjunction of P and Q if no external event has been observed yet, or like their disjunction if the decision has been made. In a timed environment, the action in fact requires that any waiting trace must be agreed by both actions. For example, $Wait\ 2 \,\Box\, Wait\ 3 = Wait\ 2$ should hold because the waiting observation after two time units cannot be satisfied by $Wait\ 2$. The semantics of the external choice in *Circus Time* is as follows:

$$P \,\Box\, Q \mathrel{\widehat{=}} \mathbf{CSP2_t}(ExtChoice1(P, Q) \vee ExtChoice2(P, Q)) \tag{2.28}$$
$$ExtChoice1(P, Q) \mathrel{\widehat{=}} P \wedge Q \wedge Stop \tag{2.29}$$
$$ExtChoice2(P, Q) \mathrel{\widehat{=}} DifDetected(P, Q) \wedge (P \vee Q) \tag{2.30}$$

$$DifDetected(P, Q) \mathrel{\widehat{=}} \tag{2.31}$$
$$\left(\neg\, ok' \vee \left(ok \wedge \neg\, wait \wedge \left(\left(\begin{array}{l} P \wedge Q \wedge ok' \wedge \\ wait' \wedge trace' = \langle\rangle \end{array}\right) \vee Skip\right)\right)\,;\, \left(\begin{array}{l} (ok' \wedge \neg\, wait' \wedge tr'_t = tr_t) \vee \\ (ok' \wedge fst(head(dif(tr'_t, tr_t))) \neq \langle\rangle \end{array}\right)\right)$$

Of course, $DifDetected(P, Q)$ is used to determine which action should be chosen and also discard the other action. Simply speaking, $\neg\, ok'$ captures the behaviour in which either P or Q diverges, the front part of the sequential composition captures the observations in which either P and Q do agree on waiting or they do not agree on waiting at all (by *Skip*), and the back part states that the agreement of P and Q is followed by either termination without changes or termination with communication.

Internal Choice

The internal choice is completely out of control of its environment. Therefore, identical to the definition in the UTP, it is simply defined as follows:

$$P \sqcap Q \mathrel{\widehat{=}} P \vee Q \tag{2.32}$$

Parallel Composition

The parallel composition $P \,|[\, s_1 \mid \{\!|\ CS\ |\!\} \mid s_2 \,]|\, Q$ is the action where all events in the set CS must be synchronised, and the events outside CS can execute independently. In addition, s_1 and s_2 contain the local variables that each action can change during the composition. The parallel process terminates only if both P and Q terminate, and it becomes divergent if either of P and Q does so.

The parallel composition of two actions in *Circus Time* is constructed as the same approach in the UTP. That is, it first transforms two actions into disjoint actions by renaming their alphabets, executes them by arbitrary interleaving and generates the final result by a merge function.

$$P \,|[\, s_1 \mid \{\!|\ CS\ |\!\} \mid s_2 \,]|\, Q \mathrel{\widehat=} (P \,;\, U0(out\alpha P) \wedge (Q \,;\, U1(out\alpha Q))_{+\{tr_t, state\}} \,;\, M_{\|} \tag{2.33}$$

The labelling process $Ul(m)$ simply passes dashed variables of its predecessor to labelled variables, which is defined as

$$Ul(m) \mathrel{\widehat=} \mathbf{var}\ l.m \,;\, (l.m := m) \,;\, \mathbf{end}\ m$$

where further

$$\mathbf{var}\ x \mathrel{\widehat=} \exists\, x \bullet \mathit{I\!I}_A \qquad \mathbf{end}\ x \mathrel{\widehat=} \exists\, x' \bullet \mathit{I\!I}_A$$

Notice that $\mathit{I\!I}_A$ is simply a relational identity ($\alpha \mathit{I\!I}_A = A$) and x is also included in A. Therefore, through the variable declaration **var** and the variable undeclaration **end** operators, the alphabet of $Ul(m)$ consists of m and $l.m$. Unfortunately, $Ul(m)$ only obtains the values of dashed variables of its predecessor. However under some circumstances we do need the initial values of its predecessor's variables. For this reason, we expand the alphabet after the labelling process. For example, $P_{+\{n\}}$ denotes $P \wedge n' = n$. Here we are only interested in tr_t and *state* that will be used in $M_{\|}$.

The function $M_{\|}$ merges timed traces from its predecessor in respect to the interface CS, and also updates the variables from s_1 and s_2, defined as follows:

$$M_{\|} \mathrel{\widehat=} M_t \,;\, \mathit{Skip} \tag{2.34}$$

$$M_t \mathrel{\widehat=} \left(\begin{array}{c} ok' = (0.ok \wedge 1.ok) \wedge wait' = (0.wait \vee 1.wait) \wedge state' = (0.state - s_2) \oplus (1.state - s_1) \\ \wedge\ dif(tr_t', tr_t) \in TSync(dif(0.tr_t, tr_t), dif(1.tr_t, tr_t), CS) \end{array} \right) \tag{2.35}$$

where the synchronisation function *TSync* takes two timed traces and a set of events on which the actions should synchronise, and returns the set containing all possible timed traces. Given two timed traces S_1 and S_2, *TSync* is defined as follows:

$$\begin{aligned}
TSync(S_1, S_2, CS) &= TSync(S_2, S_1, CS) \\
TSync(\langle\rangle, \langle\rangle, CS) &= \{\} \\
TSync(\langle(t, r)\rangle, \langle\rangle, CS) &= \{\langle(t', r)\rangle \mid t' \in Sync(t, \langle\rangle, CS)\} \\
TSync\left(\begin{array}{l} \langle(t_1, r_1)\rangle \frown S_1, \\ \langle(t_2, r_2)\rangle \frown S_2, \\ CS \end{array} \right) &= \left\{ \begin{array}{l} \langle(t', r')\rangle \mid t' \in Sync(t_1, t_2, CS) \wedge \\ r' = \left(\begin{array}{l} ((r_1 \cup t_2) \cap CS) \cup \\ ((r_1 \cap r_2) \setminus CS) \end{array} \right) \end{array} \right\} \frown TSync(S_1, S_2, CS)
\end{aligned}$$

where *Sync* is defined for the synchronisation of two sequences of events,

$$\begin{aligned}
Sync(t_1, t_2, CS) &= Sync(t_2, t_1, CS) \\
Sync(\langle\rangle, \langle\rangle, CS) &= \{\langle\rangle\} \\
Sync(\langle e_1\rangle \frown t, \langle\rangle, CS) &= \{\} \ \ \mathit{iff}\ \ e_1 \in CS \\
Sync(\langle e_1\rangle \frown t, \langle\rangle, CS) &= \{\langle e_1\rangle\} \frown Sync(t, \langle\rangle, CS) \ \ \mathit{iff}\ \ e_1 \notin CS \\
Sync(\langle e_1\rangle \frown t_1, \langle e_2\rangle \frown t_2, CS) &= \left(\begin{array}{l} \{\langle e_1\rangle\} \frown Sync(t_1, \langle e_2\rangle \frown t_2, CS) \cup \\ \{\langle e_2\rangle\} \frown Sync(\langle e_1\rangle \frown t_1, t_2, CS) \end{array} \right) \ \ \mathit{iff}\ \ e_1 \notin CS \wedge e_2 \notin CS \\
Sync(\langle e_1\rangle \frown t_1, \langle e_2\rangle \frown t_2, CS) &= \{\langle e_1\rangle\} \frown Sync(t_1, \langle e_2\rangle \frown t_2, CS) \ \mathit{iff}\ \ e_1 \notin CS \wedge e_2 \in CS \\
Sync(\langle e_1\rangle \frown t_1, \langle e_1\rangle \frown t_2, CS) &= \{\langle e_1\rangle\} \frown Sync(t_1, t_2, CS) \ \mathit{iff}\ \ e_1 \in CS \\
Sync(\langle e_1\rangle \frown t_1, \langle e_2\rangle \frown t_2, CS) &= \{\} \ \mathit{iff}\ \ e_1 \in CS \wedge e_2 \in CS \wedge e_1 \neq e_2
\end{aligned}$$

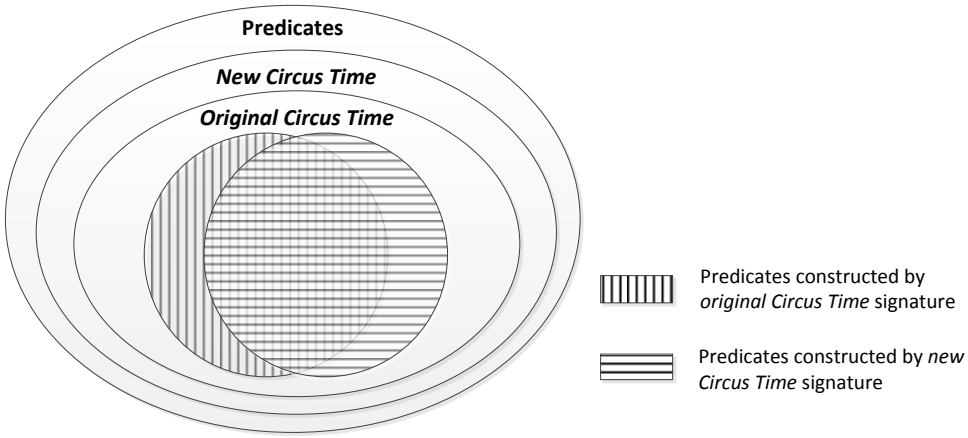

Figure 2: The relation of the new and original *Circus Time* theories

Hiding

The action $P \setminus CS$ will pass through the same performance with P, but the events in the set CS become invisible. It is defined as follows:

$$P \setminus CS \mathrel{\widehat{=}} \mathbf{R_t}(\exists\, s_t \bullet P[s_t/tr'_t] \wedge dif(tr'_t, tr_t) = dif(s_t, tr_t) \downarrow_t (Event - CS))\,;\, Skip \tag{2.36}$$

$$t_b = t_a \downarrow_t CS \Leftrightarrow \forall\, i : 1..\#t_a \bullet \begin{pmatrix} fst(t_b(i)) = fst(t_a(i)) \downarrow CS \wedge \#t_a = \#t_b \wedge \\ snd(t_a(i)) = (snd(t_b(i)) \cup (Event - CS) \end{pmatrix} \tag{2.37}$$

where $t_a \downarrow_t CS$ returns a timed trace whose events are in CS.

Recursion

The recursive action $\mu\, X \bullet P$ behaves like P with every occurrence of the system variable X in P representing a recursive invocation. For example, to express a simple recursive action $P = a \rightarrow P$, we have a monotonic function F, a variable X, and an equation $F(X) = a \rightarrow X$; and then P is actually represented by $\mu\, X.F(X)$ which stands for the least fixed point of the above equation.

Timeout

The time-sensitive choice $P \triangleright\{d\}\ Q$ resolves the choice in favour of P if P is able to execute observable events by the time d, or resolves the choice in favour of Q.

$$P \triangleright\{d\}\ Q = (P \,\Box\, (Wait\ d\,;\, int \rightarrow Q)) \setminus \{int\} \qquad (int \notin \alpha A)$$

3 New *Circus Time* Action Model

As a timed version of *Circus*, *Circus Time*, here, only concentrates on the CSP constructs since the theory of integration with other specifications like Z specifications is simply the same as that in *Circus*. Therefore, in this report we develop a new UTP theory to enable the use of pre and postconditions for reasoning in the context of *Circus Time*. Also, we use *Circus Time* to denote the new *Circus Time* mode from now on, and the previous model is always qualified as the original model whenever it is mentioned.

A theory in the UTP is a collection of relations (or alphabetised predicates), which contains three essential parts: an alphabet, a signature, and healthiness conditions. As the theories that we have introduced in previous sections, the alphabet is a set of variable names for observation, the signature gives a set of operators and atomic components of the programming theory, and the healthiness conditions identify properties that characterise the theory. The *Circus Time* theory is based on the existing semantics of the original *Circus Time* theory, as it has the same alphabet as that in the original theory, and has nearly same healthiness

conditions but $\mathbf{R2_t}$, and contains more operators and primitive actions. From the viewpoint of the UTP, the relation of the new and original *Circus Time* theories is illustrated by Figure 2.

The healthiness conditions in both theories are isomorphic except for $\mathbf{R2_t}$ from which we remove the refusals. However, $\mathbf{R2_t}$ in the original theory contains both traces and refusals. Therefore, we, here, have a weaker $\mathbf{R2_t}$. And this is the reason why the new theory is the outer oval in Figure 2. The new theory has a simper mathematical model that can get rid of much burden on proofs. In addition, this weaker $\mathbf{R2_t}$ does not affect any refinement laws of the existing operators in the original theory. That is, any operator in the original theory can also satisfy this new **R2**.

Need to think about it.

Predicates constructed by the signature of a theory may be a smaller set than the collection that satisfies the healthiness conditions. For example, the strongest predicate, *Miracle*, has been contained in the original theory, but can never be generated by using the operators in its syntax. The signature of the new theory includes all operators in the original theory. We, however, define more restricted operators, such as prefix, external choice and parallelism, by weakening their preconditions and strengthening the postconditions. That is the reason why the circle filled with vertical lines is not fully included by the circle filled with horizontal lines. These more rigorous operators can refine those in the original theory, even if they have applied a weaker **R2**. This refinement relation can validate these newly-established definitions. The proof for the refinement can also be found in Section **HERE**.

3.1 Observation

In *Circus Time*, a CSP action is described as an alphabetised predicate whose observational variables include ok, $wait$, tr, ref, $state$ and their dashed counterparts. Here, ok, ok', $wait$ and $wait'$ are the same variables used in the theory of reactive processes. The traces, tr and tr', are defined to be non-empty sequences ($\mathrm{seq}_1(\mathrm{seq}\ Event)$), and each element in the trace represents a sequence of events that have occurred over one time unit. Also, ref and ref' are non-empty sequences ($\mathrm{seq}_1(\mathbb{P}\ Event)$) where each element is a refusal at the end of a time unit. Thus, time is actually hidden in the length of traces. In addition, $state$ and $state'$ ($N \pfun Value$) records a set of local variables and their values. N is the set of valid names of these variables.

Both the original *Circus Time* and Timed CSP use the concept of failures, each of which consists of a trace and a refusal. However, this structure in the original theory, e.g., a failure is a sequence of pairs containing a sequence of events and a refusal set, is hard to manipulate. In the new *Circus Time* theory, we split a failure as shown above to record traces and refusals respectively, and use their indices (which start at 1) to match related pairs. Although we obtain a simpler mathematical model in pattern for expressing behaviours, the decomposition of the failures results in a little bit of inconvenience, since we have to ensure the equality of the lengths of tr and ref, or tr' and ref'. This is achieved by imposing an extra constraint as a healthiness condition. In addition, rather than recording the refusals only at the end of traces in CSP, *Circus Time* records the refusals at the end of each time unit. That is, we need to keep the history of refusals.

Additionally, an expanding relation between traces only in *Circus Time* is defined as follows, requiring that $front(tr)$ and $last(tr)$ are the prefixes of tr' and $tr'(\#tr)$ respectively. This corresponds to the the predicate *Expands* in Section 2.6.1.

$$tr \preccurlyeq tr' \mathrel{\widehat=} front(tr) \leq tr' \land last(tr) \leq tr'(\#tr) \tag{3.38}$$

3.2 Healthiness conditions

The *Circus Time* theory has the nearly same healthiness conditions but **R2** as those in the original theory, which are also similar to those of the CSP theory, except for some necessary changes to accommodate time. To make it visually different from those in any other theories, we annotate these healthiness condition with a subscription.

$$\mathbf{R1_{ct}}(P) \mathrel{\widehat=} P \land RT \qquad \mathbf{R2_{ct}}(P(tr, tr')) \mathrel{\widehat=} P(\langle\rangle, diff(tr', tr))$$
$$\mathbf{R3_{ct}}(P) \mathrel{\widehat=} \mathit{II}_{ct} \lhd wait \rhd P \qquad \mathbf{CSP1_{ct}}(P) \mathrel{\widehat=} P \lor (\neg\ ok \land RT)$$
$$\mathbf{CSP2_{ct}}(P) \mathrel{\widehat=} P\ ;\ J_{ct} \qquad \mathbf{CSP3_{ct}}(P) \mathrel{\widehat=} Skip\ ;\ P$$
$$\mathbf{CSP4_{ct}}(P) \mathrel{\widehat=} P\ ;\ Skip \qquad \mathbf{CSP5_{ct}}(P) \mathrel{\widehat=} P \ |||\ Skip$$

Here, the predicate RT, the functions *diff*, the identity $\mathbb{I}_{ct}$ and J_{ct} are given as

$$RT \mathrel{\widehat{=}} tr \preccurlyeq tr' \wedge front(ref) \leq ref' \wedge \#tr = \#ref \wedge \#tr' = \#ref' \tag{3.39}$$

$$diff(tr', tr) \mathrel{\widehat{=}} \langle tr'(\#tr) - last(tr) \rangle \frown tail(tr' - front(tr)) \tag{3.40}$$

$$\mathbb{I}_{ct} \mathrel{\widehat{=}} (\neg\, ok \wedge RT) \vee (ok' \wedge \mathbb{I} \wedge \#tr = \#ref \wedge \#tr' = \#ref') \tag{3.41}$$

$$J_{ct} \mathrel{\widehat{=}} (ok \Rightarrow ok') \wedge (tr' = tr \wedge ref' = ref \wedge wait' = wait \wedge state' = state) \tag{3.42}$$

We use *diff* to obtain the difference between two traces, and also redefine the identity $\mathbb{I}_{ct}$, and J_{ct} in the *Circus Time* theory. Because we record the whole history of refusals, we impose $front(ref) \leq ref'$ in $\mathbf{R1_{ct}}$ to ensure that an action can never change the history of both traces and refusals. However, we are usually not interested in the refusals of the last time unit after an action terminates. Therefore, we use $front(ref) \leq ref'$ rather than $ref \leq ref'$ to make this requirement get along with $\mathbf{CSP3_{ct}}$ and $\mathbf{CSP4_{ct}}$. Also, we impose a restriction, $\#diff(tr', tr) = \#(ref' - front(ref))$, to ensure that the lengths of *ref* and ref' are always the same as those of *tr* and tr' respectively. This is a consequence of splitting traces and refusals as already explained. Contrary to *diff*, we define a new concatenation operation (a reverse operation to *diff*) on traces in *Circus Time*, and its properties are given in Appendix A.1.

$$conc(s, t) \mathrel{\widehat{=}} front(s) \frown \langle last(s) \frown first(t) \rangle \frown tail(t) \tag{3.43}$$

In *Circus Time*, we have a simpler $\mathbf{R2_{ct}}$, which requires that only the previous trace has no influence on the behaviour of an action. This simplification is inspired not only by our new approach to treat traces and refusals individually, but also by the fact that only few operators can be affected by the simpler $\mathbf{R2_{ct}}$. Here, through banning the involvement of *tr* and *ref* within these operators, for example, we require that the condition *b* does not contain *tr* and *ref* in a conditional choice, we enable the new $\mathbf{R2_{ct}}$ to be compatible with the definitions of all operators in the original theory. This compatibility is important in proving that some operators that we have made changes in the new theory can refine those in the original theory. In addition, this simpler $\mathbf{R2_{ct}}$ can simplify proofs of properties of the new theory to a certain extent.

A predicate is a *Circus Time* action only if it is a fixed point of the composition of $\mathbf{CSP1_{ct}}$-$\mathbf{CSP5_{ct}}$ and $\mathbf{R_{ct}}$ where $\mathbf{R_{ct}} \mathrel{\widehat{=}} \mathbf{R1_{ct}} \circ \mathbf{R2_{ct}} \circ \mathbf{R3_{ct}}$. The healthiness conditions in *Circus Time* preserve all properties of the healthiness conditions in the CSP theory. Some selective laws for these healthiness conditions are given as follows, and their proofs can be in Appendix A.

Law 2. ($\mathbf{R1_{ct}}$-*idempotent*) $\mathbf{R1_{ct}} \circ \mathbf{R1_{ct}}(P) = \mathbf{R1_{ct}}(P)$

Law 3. (closure-$\wedge$-$\mathbf{R1_{ct}}$) $\mathbf{R1_{ct}}(P \wedge Q) = \mathbf{R1_{ct}}(P) \wedge \mathbf{R1_{ct}}(Q)$

Law 4. (closure-$\vee$-$\mathbf{R1_{ct}}$ $\mathbf{R1_{ct}}(P \vee Q) = \mathbf{R1_{ct}}(P) \vee \mathbf{R1_{ct}}(Q)$

Law 5. ($\mathbb{I}_{ct}$-$\mathbf{R1_{ct}}$) $\mathbf{R1_{ct}}(\mathbb{I}_{ct}) = \mathbb{I}_{ct}$

Law 6. (closure-; -$\mathbf{R1_{ct}}$) $\mathbf{R1_{ct}}(P;\ Q) = P;\ Q$ *provided* P *and* Q *are* $\mathbf{R1_{ct}}$ *healthy*

Law 7. (closure-$\lhd\rhd$-$\mathbf{R1_{ct}}$) $\mathbf{R1_{ct}}(P \lhd b \rhd Q) = \mathbf{R1_{ct}}(P) \lhd b \rhd \mathbf{R1_{ct}}(Q)$

Law 8. ($\mathbf{R2_{ct}}$-*idempotent*) $\mathbf{R2_{ct}} \circ \mathbf{R2_{ct}}(P) = \mathbf{R2_{ct}}(P)$

Law 9. (closure-$\wedge$-$\mathbf{R2_{ct}}$) $\mathbf{R2_{ct}}(P \wedge Q) = \mathbf{R2_{ct}}(P) \wedge \mathbf{R2_{ct}}(Q)$

Law 10. (closure-$\vee$-$\mathbf{R2_{ct}}$) $\mathbf{R2_{ct}}(P \vee Q) = \mathbf{R2_{ct}}(P) \vee \mathbf{R2_{ct}}(Q)$

Law 11. ($\mathbb{I}_{ct}$-$\mathbf{R2_{ct}}$) $\mathbf{R2_{ct}}(\mathbb{I}_{ct}) = \mathbb{I}_{ct}$

Law 12. (closure-; -$\mathbf{R2_{ct}}$) $\mathbf{R2_{ct}}(P;\ Q) = P;\ Q$ *provided* P *and* Q *are* $\mathbf{R2_{ct}}$ *healthy*

Law 13. (closure-$\lhd\rhd$-$\mathbf{R2_{ct}}$) $\mathbf{R2_{ct}}(P \lhd b \rhd Q) = \mathbf{R2_{ct}}(P) \lhd b \rhd \mathbf{R2_{ct}}(Q)$ *if* b *not contain* tr *and* tr'

Law 14. (commutativity-$\mathbf{R1_{ct}}$-$\mathbf{R2_{ct}}$) $\mathbf{R2_{ct}} \circ \mathbf{R1_{ct}} = \mathbf{R1_{ct}} \circ \mathbf{R2_{ct}}$

Law 15. ($\mathbf{R3_{ct}}$-*idempotent*) $\mathbf{R3_{ct}} \circ \mathbf{R3_{ct}}(P) = \mathbf{R3_{ct}}(P)$

Law 16. (closure-$\wedge$-$\mathbf{R3_{ct}}$) $\mathbf{R3_{ct}}(P \wedge Q) = \mathbf{R3_{ct}}(P) \wedge \mathbf{R3_{ct}}(Q)$

Law 17. (closure-$\vee$-$\mathbf{R3_{ct}}$) $\mathbf{R3_{ct}}(P \vee Q) = \mathbf{R3_{ct}}(P) \vee \mathbf{R3_{ct}}(Q)$

Law 18. ($\mathbb{I}_{ct}$-$\mathbf{R3_{ct}}$) $\mathbf{R3_{ct}}(\mathbb{I}_{ct}) = \mathbb{I}_{ct}$

Law 19. (closure-; -$\mathbf{R3_{ct}}$) $\mathbf{R3_{ct}}(P;\ Q) = P;\ Q$ *provided* P *is* $\mathbf{R3_{ct}}$ *and* Q *is* $\mathbf{R1_{ct}}$ *and* $\mathbf{R3_{ct}}$

Law 20. (closure-$\lhd\rhd$-$\mathbf{R3_{ct}}$) $\mathbf{R3_{ct}}(P \lhd b \rhd Q) = \mathbf{R3_{ct}}(P) \lhd b \rhd \mathbf{R3_{ct}}(Q)$

Law 21. (commutativity-$\mathbf{R1_{ct}}$-$\mathbf{R3_{ct}}$) $\mathbf{R3_{ct}} \circ \mathbf{R1_{ct}} = \mathbf{R1_{ct}} \circ \mathbf{R3_{ct}}$

Law 22. (commutativity-$\mathbf{R2_{ct}}$-$\mathbf{R3_{ct}}$) $\mathbf{R3_{ct}} \circ \mathbf{R2_{ct}} = \mathbf{R2_{ct}} \circ \mathbf{R3_{ct}}$

Law 23. ($\mathbf{R3_{ct}}$-ok') $(\mathbf{R3_{ct}}(P))^c = ((\mathbb{I}_{ct})^c \lhd \mathit{wait} \rhd P^c)$

Law 24. ($\mathbf{R_{ct}}$-*wait*-**false**) $(\mathbf{R_{ct}}(P))_f = \mathbf{R1_{ct}} \circ \mathbf{R2_{ct}}(P_f)$

Law 25. ($\mathbf{R_{ct}}$-*wait*-**true**) $(\mathbf{R_{ct}}(P))_t = (\mathbb{I}_{ct})_t$

Law 26. ($\mathbf{CSP1_{ct}}$-*idempotent*) $\mathbf{CSP1_{ct}} \circ \mathbf{CSP1_{ct}} = \mathbf{CSP1_{ct}}$

Law 27. (commutativity-$\mathbf{R1_{ct}}$-$\mathbf{CSP1_{ct}}$) $\mathbf{CSP1_{ct}} \circ \mathbf{R1_{ct}} = \mathbf{R1_{ct}} \circ \mathbf{CSP1_{ct}}$

Law 28. (commutativity-$\mathbf{R2_{ct}}$-$\mathbf{CSP1_{ct}}$) $\mathbf{CSP1_{ct}} \circ \mathbf{R2_{ct}} = \mathbf{R2_{ct}} \circ \mathbf{CSP1_{ct}}$

Law 29. (commutativity-$\mathbf{R3_{ct}}$-$\mathbf{CSP1_{ct}}$) $\mathbf{CSP1_{ct}} \circ \mathbf{R3_{ct}} = \mathbf{R3_{ct}} \circ \mathbf{CSP1_{ct}}$

Law 30. $\mathbf{R_{ct}} \circ \mathbf{CSP1_{ct}}(P) = \mathbf{CSP1_{ct}} \circ \mathbf{R_{ct}}(P)$

Law 31. ($\mathbf{CSP1_{ct}}$-$\mathbf{R1_{ct}}$-$\mathbf{H1}$) $\mathbf{CSP1_{ct}} = \mathbf{R1_{ct}} \circ \boldsymbol{H1}$ *provided* P *is* $\mathbf{R1_{ct}}$

Law 32. (closure-$\wedge$-$\mathbf{CSP1_{ct}}$) $\mathbf{CSP1_{ct}}(P \wedge Q) = P \wedge Q$ *provided* P *and* Q *are* $\mathbf{CSP1_{ct}}$

Law 33. (closure-$\vee$-$\mathbf{CSP1_{ct}}$) $\mathbf{CSP1_{ct}}(P \vee Q) = P \vee Q$ *provided* P *and* Q *are* $\mathbf{CSP1_{ct}}$

Law 34. (closure-$\lhd\rhd$-$\mathbf{CSP1_{ct}}$) $\mathbf{CSP1_{ct}}(P \lhd b \rhd Q) = P \lhd b \rhd Q$ *provided* P *and* Q *are* $\mathbf{CSP1_{ct}}$

Law 35. (closure-; -$\mathbf{CSP1_{ct}}$) $\mathbf{CSP1_{ct}}(P\,;\,Q) = P\,;\,Q$ *provided* P *and* Q *are* $\mathbf{CSP1_{ct}}$

Law 36. ($\mathbf{CSP2_{ct}}$-*idempotent*) $\mathbf{CSP2_{ct}} \circ \mathbf{CSP2_{ct}} = \mathbf{CSP2_{ct}}$

Law 37. (closure-$\vee$-$\mathbf{CSP2_{ct}}$) $\mathbf{CSP2_{ct}}(P \vee Q) = P \vee Q$ *provided* P *and* Q *are* $\mathbf{CSP2_{ct}}$

Law 38. (closure-$\lhd\rhd$-$\mathbf{CSP2_{ct}}$) $\mathbf{CSP2_{ct}}(P \lhd b \rhd Q) = P \lhd b \rhd Q$ *provided* P *and* Q *are* $\mathbf{CSP2_{ct}}$

Law 39. (closure-; -$\mathbf{CSP2_{ct}}$) $\mathbf{CSP2_{ct}}(P\,;\,Q) = P\,;\,Q$ *provided* Q *is* $\mathbf{CSP2_{ct}}$

Law 40. (J-splitting) $P\,;\,J = P^f \vee (P^t \wedge ok')$

Moreover, similar to the result in the CSP theory, each action in the *Circus Time* theory can be described as a reactive design.

Theorem 1. *(***Reactive design***) For every action* P *in Circus Time,*

$$P = \mathbf{R_{ct}}(\neg\ P_f^f \vdash P_f^t)$$

Proof.

$$
\begin{array}{ll}
& P \\
= & \mathbf{CSP2_{ct}} \circ \mathbf{CSP1_{ct}} \circ \mathbf{R_{ct}}(P) & \text{[Law 30]} \\
= & \mathbf{CSP2_{ct}} \circ \mathbf{R_{ct}} \circ \mathbf{CSP1_{ct}}(P) & \text{[Law 31]} \\
= & \mathbf{CSP2_{ct}} \circ \mathbf{R_{ct}} \circ \mathbf{R1_{ct}} \circ \mathbf{H1}(P) & \text{[Law 2]} \\
= & \mathbf{CSP2_{ct}} \circ \mathbf{R_{ct}} \circ \mathbf{H1}(P) & [\mathbf{H1}] \\
= & \mathbf{CSP2_{ct}} \circ \mathbf{R_{ct}}(ok \Rightarrow P) & \text{[propositional calculus]} \\
= & \mathbf{CSP2_{ct}} \circ \mathbf{R_{ct}}(\neg\ ok \vee P) & [\mathbf{CSP2_{ct}}] \\
= & \mathbf{R_{ct}}(\neg\ ok \vee P)\,;\,J & \text{[Law 40]} \\
= & (\mathbf{R_{ct}}(\neg\ ok \vee P))^f \vee ((\mathbf{R_{ct}}(\neg\ ok \vee P))^t \wedge ok') & \text{[Law 15]} \\
= & (\mathbf{R3_{ct}} \circ \mathbf{R_{ct}}(\neg\ ok \vee P))^f \vee ((\mathbf{R3_{ct}} \circ \mathbf{R_{ct}}(\neg\ ok \vee P))^t \wedge ok') & \text{[Law 23]} \\
= & (\mathbb{I}_{ct})^f \lhd \mathit{wait} \rhd (\mathbf{R_{ct}}(\neg\ ok \vee P))^f \vee ((\mathbb{I}_{ct})^t \lhd \mathit{wait} \rhd (\mathbf{R_{ct}}(\neg\ ok \vee P))^t \wedge ok') & \text{[relational calculus]} \\
= & (((\mathbb{I}_{ct})^f \vee ((\mathbb{I}_{ct})^t \wedge ok')) \lhd \mathit{wait} \rhd ((\mathbf{R_{ct}}(\neg\ ok \vee P))^f \vee ((\mathbf{R_{ct}}(\neg\ ok \vee P))^t \wedge ok'))) & [\mathbb{I}_{ct} \text{ and subs.}]
\end{array}
$$

$$
\begin{array}{ll}
= \mathit{II}_{ct} \lhd \mathit{wait} \rhd (\mathbf{R_{ct}}(\neg\ ok \vee P))^f \vee ((\mathbf{R_{ct}}(\neg\ ok \vee P))^t \wedge ok') & \text{[case split on } \mathit{wait} \text{ in the right]} \\
= \mathit{II}_{ct} \lhd \mathit{wait} \rhd \left(\begin{array}{c} (\mathbf{R_{ct}}(\neg\ ok \vee P))^f_f \vee (\mathbf{R_{ct}}(\neg\ ok \vee P))^f_t \vee \\ ((\mathbf{R_{ct}}(\neg\ ok \vee P))^t_f \wedge ok') \vee ((\mathbf{R_{ct}}(\neg\ ok \vee P))^t_t \wedge ok') \end{array} \right) & \text{[Law 25]} \\
= \mathit{II}_{ct} \lhd \mathit{wait} \rhd \left((\mathbf{R_{ct}}(\neg\ ok \vee P))^f_f \vee (\mathit{II}_{ct})^f_t \vee ((\mathbf{R_{ct}}(\neg\ ok \vee P))^t_f \wedge ok') \vee ((\mathit{II}_{ct})^t_t \wedge ok') \right) & \text{[Law 24]} \\
= \mathit{II}_{ct} \lhd \mathit{wait} \rhd \left(\begin{array}{c} (\mathbf{R1_{ct}} \circ \mathbf{R2_{ct}}(\neg\ ok \vee P_f))^f \vee (\mathit{II}_{ct})^f_t \vee \\ ((\mathbf{R1_{ct}} \circ \mathbf{R2_{ct}}(\neg\ ok \vee P_f))^t \wedge ok') \vee ((\mathit{II}_{ct})^t_t \wedge ok') \end{array} \right) & \text{[}\mathit{II}_{ct} \text{ and substitution]} \\
= \mathit{II}_{ct} \lhd \mathit{wait} \rhd \big((\mathbf{R1_{ct}} \circ \mathbf{R2_{ct}}(\neg\ ok \vee P_f))^f \vee (\mathit{II}_{ct})_t \vee ((\mathbf{R1_{ct}} \circ \mathbf{R2_{ct}}(\neg\ ok \vee P_f))^t \wedge ok')\big) & \text{[rel. cal.]} \\
= \mathit{II}_{ct} \lhd \mathit{wait} \rhd \big((\mathbf{R1_{ct}} \circ \mathbf{R2_{ct}}(\neg\ ok \vee P_f))^f \vee ((\mathbf{R1_{ct}} \circ \mathbf{R2_{ct}}(\neg\ ok \vee P_f))^t \wedge ok') \big) & \text{[substitution]} \\
= \mathit{II}_{ct} \lhd \mathit{wait} \rhd \left((\mathbf{R1_{ct}} \circ \mathbf{R2_{ct}}(\neg\ ok \vee P^f_f)) \vee ((\mathbf{R1_{ct}} \circ \mathbf{R2_{ct}}(\neg\ ok \vee P^t_f)) \wedge ok') \right) & \text{[Law 4,10]} \\
= \mathit{II}_{ct} \lhd \mathit{wait} \rhd (\mathbf{R1_{ct}} \circ \mathbf{R2_{ct}}(\neg\ ok \vee P^f_f \vee (P^t_f \wedge ok'))) & [\mathbf{R3_{ct}}] \\
= \mathbf{R_{ct}}(\neg\ ok \vee P^f_f \vee (P^t_f \wedge ok')) & \text{[design]} \\
= \mathbf{R_{ct}}(\neg\ P^f_f \vdash P^t_f) &
\end{array}
$$

□

3.3 Syntax

A *Circus* program defines a collection of processes that consist of states defined using Z (usually as schemas), and behaviours defined using a mixture of Z and CSP constructs as actions. Processes can also be defined using CSP constructs only. Here, *Circus Time* focuses on the notation of actions only defined by CSP constructs. As shown in Figure 3, the signature of the *Circus Time* theory contains more operators and atomic actions than that of the original *Circus Time* theory. Figure 3 presents the description of the syntax

$$
\begin{array}{l}
P ::= \mathit{Skip} \mid \mathit{Stop} \mid \mathit{Miracle} \mid \mathit{Chaos} \mid N := e \mid c.e \rightarrow P \mid P \,;\, Q \mid P \lhd b \rhd Q \mid b \& P \mid \\
\quad P \,\Box\, Q \mid P \sqcap Q \mid P \,|[\, s_1 \mid \{\!|\, CS \,|\!\} \mid s_2 \,]|\, Q \mid P \setminus CS \mid \mathit{Wait}\ d \mid \mathit{Wait}\ d_1..d_2 \mid \\
\quad \mu\, N \bullet P \mid P \rhd \{d\}\ Q \mid P \blacktriangleright d \mid d \blacktriangleleft P \mid c.e@t \rightarrow P
\end{array}
$$

Figure 3: *Circus Time* Syntax

of *Circus Time*. In this figure, b stands for a condition, c for a communication channel, e for any expression, d, d_1 and d_2 for integer numbers, t is a time variable, N for any valid name (or identifier), s for a set of variable names, and CS for a set of channel names.

Most primitive actions and operators are the same as those in Timed CSP. *Skip* is a primitive action that terminates immediately without changing anything. *Stop* represents deadlock, but allows time to elapse. *Chaos* is the worst action and its behaviour is unpredictable. The assignment action (:=) simply assigns a value to a state variable and other state variables keep unchanged. This action does not take any time. The delay action *Wait* d does nothing except that it allows d time units to pass before it terminates.

The sequential composition $P\,;\,Q$ behaves like P until P terminates, and then behaves like Q. The prefix action $c.e \rightarrow P$ is able to pass the value e through the channel c and then behaves as P. This action can also be expressed by a composition of a simple prefix and P itself, written as $(c.e \rightarrow \mathit{Skip})\,;\,P$. The conditional choice $P \lhd b \rhd Q$ behaves like P if b is true, otherwise behaves like Q. The guarded action $b\&P$ behaves like P only if b is true, and therefore it can also be expressed as a conditional choice as $P \lhd b \rhd \mathit{Stop}$.

The external choice $P \,\Box\, Q$ may behave either like the conjunction of P and Q if no external event has been observed yet, or like their disjunction if the decision has been made. The internal choice $P \sqcap Q$ is completely out of control of its environment, and behaves like P or Q arbitrarily. The parallel composition $P \,|[\, s_1 \mid \{\!|\, CS \,|\!\} \mid s_2 \,]|\, Q$ is the action where all events in the set CS must be synchronised, and the events outside CS can execute independently. In addition, s_1 and s_2 contain the local variables that each action can change during the composition. The hiding action $P \setminus CS$ will behave like P, but the events in the set CS become invisible. The recursive action $\mu\, X \bullet P$ behaves like P with every occurrence of the variable X in P

representing a recursive invocation. The recursive call takes no time. The time-sensitive choice $P \triangleright\{d\}\ Q$ resolves the choice in favour of P if P is able to execute observable events by the time d, or resolves the choice in favour of Q.

By comparison with the original *Circus Time*, *Circus Time* introduces some new operators. Some of them have been previously defined in Timed CSP such as timed event prefix ($c.e@t \rightarrow P$) and nondeterministic delay ($Wait\ d_1..d_2$). The nondeterministic delay can wait for any time unit whose value is between d_1 and d_2. A timed event prefix allows the action to record the amount of time which has elapsed between the initial event's offer and its occurrence. The information of time may then be used by the subsequent action as a local variable.

The primitive action, *Miracle*, expresses an unstarted action. This action has been included in the semantics of the original *Circus Time*, but is ignored as an infeasible action. *Miracle* is usually considered a useless action in engineering practice because of its infeasibility, but it is so useful as a mathematical abstraction in reasoning about programs. As our best knowledge, the work in [24] firstly explores the application of *Miracle* in system specifications, and here we explore it further by using it to define two deadline operators. One ($P \blacktriangleright d$) is that an action must terminate within a deadline, the other ($d \blacktriangleleft P$) is that a visible interaction of the action must start within the deadline. Here, the deadline operators are *hard*, which means that these deadlines *must* be satisfied. For example, in the following action,

$$P = ((a \rightarrow Skip) \blacktriangleright 2) \,\Box\, (Wait\ 3)\,;(b \rightarrow Skip))$$

where b will never occur since a must happen within 2 time units and therefore always resolves the external choice prior to b.

3.4 Transformation

UTP provides linking theories to explore the relation of different models that may have different expressive power. One of the possible benefits of using links between theories in the UTP is that it is possible to create a solid semantics based on the sound semantics of a matured model.

The new *Circus Time* model developed in this report is a conservative[1] extension to the original *Circus Time* model, and therefore all the laws about healthiness conditions and various operators in the original *Circus Time* still hold in the new *Circus Time*. As a matter of fact, the two models have the nearly same semantic model because they have the same healthiness conditions except for $\mathbf{R2_{ct}}$. However, all operators in the original *Circus Time* satisfy the new $\mathbf{R2_{ct}}$, which will be explained during a transformation. Even the brand-new action, *Miracle*, in the new *Circus Time* has been always existed in the original *Circus Time*, but not been explored yet. In other words, the new *Circus Time* discovers some interesting actions which are never mentioned in the original theory.

Because we have split a failure (timed trace) in the original theory into a trace and a sequence of refusals individually, we will apply this transformation to the UPT definition of each operator, which will be used to calculate its reactive design semantics later. As a result, in this section, apart from the split of failures, we also transform all healthiness conditions in the original model into those in the new model, and the UTP definition of each operator in the original theory into those with new healthiness conditions, traces and refusals.

In general, the most used way of defining a link between theories is as a function L which maps all predicates of a theory into a subset of the predicates from the other. Similarly, we define a function L which maps all the healthiness conditions and operators of the original *Circus Time* into the corresponding ones in the new *Circus Time*, and hence we are able to prove that all healthiness conditions and corresponding operators in both models are equivalent.

Suppose that actions in the original and new *Circus Time* models are expressed as $S(a)$ and $C(b)$ respectively where a and b are sets of variables. By means of a mapping function $f(a, b)$, L is defined as follows:

$$L(S(a)) \mathrel{\widehat{=}} \exists\, a \bullet S(a) \wedge f(a, b)$$

where $f(a, b)$ is defined as follows:

$$f(tr'_t, tr_t, tr', tr, ref', ref) \mathrel{\widehat{=}} \begin{pmatrix} tr = proj\_fst(tr_t) \wedge tr' = proj\_fst(tr'_t) \wedge \\ ref = proj\_snd(tr_t) \wedge ref' = proj\_snd(tr'_t) \end{pmatrix} \tag{3.44}$$

[1] In mathematical logic, a theory T_2 is a conservative extension of a theory T_1 if every theorem of T_1 is a theorem of T_2, and any theorem of T_2 which is in the language of T_1 is already a theorem of T_1.

and the functions *proj_fst* and *proj_snd* are used to split up traces and refusals of a timed trace.

$$tr = proj\_fst(tr_t) \Leftrightarrow \forall\, i : 1..\#tr_t \bullet tr(i) = fst(tr_t(i)) \land \#tr = \#tr_t \tag{3.45}$$
$$ref = proj\_snd(tr_t) \Leftrightarrow \forall\, i : 1..\#tr_t \bullet ref(i) = snd(tr_t(i)) \land \#ref = \#tr_t \tag{3.46}$$

Contrarily, we define an operator to merge *tr* and *ref* into a timed trace.

$$tr \circledast ref = tr_t \Leftrightarrow \forall\, i : 1..\#tr_t \bullet tr(i) = fst(tr_t(i)) \land ref(i) = snd(tr_t(i)) \land \#tr = \#tr_t = \#ref \tag{3.47}$$

And two laws about the split and merge operators can be easily proved.

Law 41. $tr = proj\_fst(tr \circledast ref)$

Law 42. $ref = proj\_snd(tr \circledast ref)$

An important feature of L is that, for any action X, $L(X)$ does not change the semantics of X except for transforming a timed trace of X into a trace and a refusal respectively. In the following proof, we show two important laws which consider the two special predicates, *true* and *false*.

Law 43.

$$L(\mathbf{true}) = \mathbf{true}$$

Proof.

$$\begin{aligned}
& L(\mathbf{true}) && \text{[def of } L\text{]}\\
=& \exists\, tr_t, tr_t' \bullet \begin{pmatrix} tr = proj_fst(tr_t) \land tr' = proj_fst(tr_t') \land \\ ref = proj_snd(tr_t) \land ref' = proj_snd(tr_t') \end{pmatrix} \land \mathbf{true} && \text{[propositional calculus]}\\
=& \exists\, tr_t, tr_t' \bullet (tr = proj_fst(tr_t) \land tr' = proj_fst(tr_t') \land ref = proj_snd(tr_t) \land ref' = proj_snd(tr_t')) \\
& && \text{[Let } tr_t = \langle (tr, ref) \rangle \text{ and } tr_t' = \langle (tr', ref') \rangle\text{]}\\
\Leftarrow& \begin{pmatrix} tr = proj_fst(\langle (tr, ref) \rangle) \land tr' = proj_fst(\langle (tr', ref') \rangle) \land \\ ref = proj_snd(\langle (tr, ref) \rangle) \land ref' = proj_snd(\langle (tr', ref') \rangle) \end{pmatrix} && \text{[properties of } proj_fst, proj_snd\text{]}\\
=& (tr = tr \land tr' = tr' \land ref = ref \land ref' = ref') && \text{[property of equality]}\\
=& \mathbf{true}
\end{aligned}$$

□

Law 44.

$$L(\mathbf{false}) = \mathbf{false}$$

Proof.

$$\begin{aligned}
& L(\mathbf{false}) && \text{[def of } L\text{]}\\
=& \exists\, tr_t, tr_t' \bullet \begin{pmatrix} tr = proj_fst(tr_t) \land tr' = proj_fst(tr_t') \land \\ ref = proj_snd(tr_t) \land ref' = proj_snd(tr_t') \end{pmatrix} \land \mathbf{false} && \text{[propositional calculus]}\\
=& \exists\, tr_t, tr_t' \bullet \mathbf{false} && \text{[predicate calculus]}\\
=& \mathbf{false}
\end{aligned}$$

□

3.4.1 Properties of L on *Circus Time*

We list some useful laws about *proj_fst* and *proj_snd*. The proofs of these laws will not be given here since they can be easily proved by applying induction theorems on sequences.

Property 2.

$$L1.\ (front(tr_t) \leq tr'_t) = (front(proj\_fst(tr_t)) \leq proj\_fst(tr'_t) \land front(proj\_snd(tr_t)) \leq proj\_snd(tr'_t))$$

$$L2.\ fst(last(tr_t)) = last(proj\_fst(tr_t))$$

$$L3.\ \forall\, i : 1..\#tr_t \bullet fst(tr_t(i)) = proj\_fst(tr_t)(i)$$

$$L4.\ \#tr_t = \#proj\_fst(tr_t) = \#(proj\_snd(tr_t))$$

$$L5.\ (tr_t = tr'_t) = (proj\_fst(tr_t) = proj\_fst(tr'_t) \land proj\_snd(tr_t) = proj\_snd(tr'_t))$$

$$L6.\ Flat(tr_t) = \frown/(proj\_fst(tr_t))$$

$$L7.\ \forall\, i : 1..\#tr_t \bullet snd(tr_t(i)) = proj\_snd(tr_t)(i)$$

$$L8.\ (front(tr_t) = front(tr'_t)) = \left(\begin{array}{c} front(proj\_fst(tr_t)) = front(proj\_fst(tr'_t)) \land \\ front(proj\_snd(tr_t)) = front(proj\_snd(tr'_t)) \end{array} \right)$$

$$L9.\ proj\_fst(tail(tr_t)) = tail(proj\_fst(tr_t))$$

$$L10.\ proj\_snd(tail(tr_t)) = tail(proj\_snd(tr_t))$$

$$L11.\ proj\_fst(front(tr_t)) = front(proj\_fst(tr_t))$$

$$L12.\ proj\_snd(front(tr_t)) = front(proj\_snd(tr_t))$$

$$L13.\ \forall\, i : 1..\#tr_t \bullet fst(tr_t(i)) = proj\_fst(tr_t)(i)$$

$$L14.\ (\langle\rangle \leq tr_t) = (\langle\rangle \leq proj\_fst(tr_t) \land \langle\rangle \leq proj\_snd(tr_t)$$

$$L15.\ proj\_fst(dif(tr'_t, tr_t)) = diff(proj\_fst(tr'_t), proj\_snd(tr_t))$$

$$L16.\ proj\_snd(dif(tr'_t, tr_t)) = proj\_snd(tr'_t) - front(proj\_snd(tr_t))$$

$$L17.\ (\exists\, r \bullet \langle(\langle\rangle, r)\rangle = dif(tr'_t, tr_t)) = \left(\begin{array}{c} \langle\langle\rangle\rangle = diff(proj\_fst(tr'_t), proj\_fst(tr_t)) \land \\ front(proj\_snd(tr'_t)) = front(proj\_snd(tr_t)) \end{array} \right)$$

The major purpose of the linking theory in this report is to underpin the soundness of the semantics of the new *Circus Time* from the well-built semantics of the original *Circus Time*. As a result, the mapping function L is used to prove that the semantics of primitive actions and operators in the original *Circus Time* is equal to those in the new *Circus Time* except for the different structure.

Before we further explore the relation of the semantics of the two models by means of L, we first give some useful algebraic laws where the subscripts, t and ct, are used to denote a predicate in the original and new models respectively.

Property 3.

$L1.\ L(L(X)) = L(X)$

$L2.\ L(X \lor Y) = L(X) \lor L(Y)$

$L3.\ L(X \land Y) \Rightarrow L(X) \land L(Y)$

$L4.\ L(X \land Y) = L(X) \land Y$ *provided that* tr_t *and* tr_t' *are not free in* Y

$L5.\ L(X \lhd b \rhd Y) = L(X) \lhd b \rhd L(Y)$ *provided* tr_t *and* tr_t' *are not free in* b

$L6.\ L(X\ ;\ Y) = L(X)\ ;\ L(Y)$

$L7.\ L(Expands(tr_t, tr_t')) = (tr \preccurlyeq tr' \land front(ref) \leq ref' \land \#diff(tr', tr) = \#(ref' - front(ref))$

$L8.\ L(tr_t' = tr_t) = ((tr' = tr \land ref' = ref) \land \#tr = \#ref \land \#tr' = \#ref')$

$L9.\ L(\mathit{II}_t) = \mathit{II}_{ct}$

$L10.\ L(trace' = t) = (\frown/tr' - \frown/tr = t)$

$L11.\ L(wait\_com(c)_t) = wait\_com(c)_{ct}$

$L12.\ L(term\_com(c.e)_t) = term\_com(c.e)_{ct}$

$L13.\ L(terminating\_com(c.e)_t) = terminating\_com(c.e)_{ct}$

$$L14.\ L(dif(tr_t', tr_t) \in TSync(dif(0.tr_t', tr_t), dif(1.tr_t', tr_t), CS)) = \left(\begin{array}{c} diff(tr', tr) \in SSync(diff(0.tr', tr), diff(1.tr', tr), CS) \land \\ ref' - front(ref) = MRef(0.ref' - front(ref), 1.ref' - front(ref), CS) \end{array} \right)$$

$L15.\ L(M_t) = M_{ct}$

$$L16.\ L(dif(tr_t', tr_t) = dif(s_t, tr_t) \downarrow_t (Event - CS)) = \left(\begin{array}{c} diff(tr', tr) = diff(s, tr) \downarrow_t (\Sigma - CS) \land \\ r - front(ref) = ((ref' - front(ref)) \cup_t CS \end{array} \right)$$

$L17.\ L(DifDetected(P, Q)_t) \Rightarrow DifDetected(L(P), L(Q))_{ct}$

$L18.\ L(Ul(out\alpha(P)) = Ul(out\alpha(L(P))$

The mapping function L here is very similar to that in the original *Circus Time* and hence we will not give the proofs about $L1 - L6$. The reader is referred to the original model [21] for the detailed proof. The proof about $L7 - L18$ can be found in Appendix B.1.

The function L can also map the healthiness conditions in the original *Circus Time* to those in the new model. The proofs of the following properties are in Appendix B.2.

Property 4.

$L1.\ L(\mathbf{R1_t}(X)) = \mathbf{R1_{ct}}(L(X))$

$L2.\ L(\mathbf{R2_t}(X)) \Rightarrow \mathbf{R2_{ct}}(L(X))$

$L3.\ L(\mathbf{R3_t}(X)) = \mathbf{R3_{ct}}(L(X))$

$L4.\ L(\mathbf{CSP1_t}(X)) = \mathbf{CSP1_{ct}}(L(X))$

$L5.\ L(\mathbf{CSP2_t}(X)) = \mathbf{CSP2_{ct}}(L(X))$

$L6.\ L(\mathbf{CSP3_t}(X)) = \mathbf{CSP3_{ct}}(L(X))$

$L7.\ L(\mathbf{CSP4_t}(X)) = \mathbf{CSP4_{ct}}(L(X))$

$L8.\ L(\mathbf{CSP5_t}(X)) = \mathbf{CSP5_{ct}}(L(X))$

3.4.2 Semantics of *Circus Time* after L

In this section, we give new definition to some operators, sequentially followed laws to show them equal to the corresponding definitions in the original model with the function L. These UTP definitions will be used to calculate the reactive designs in Section 4. Because the syntax of the two models are similar, we use the notation $[\![X]\!]_t$ to represent the action X in the original model, and $[\![X]\!]_{ct}$ to express the same action in the new one. Another purpose of the transformation is to show that each operator in the original *Circus Time* is $\mathbf{R2_{ct}}$ healthy even if $L(\mathbf{R2_t}(X)) \neq \mathbf{R2_{ct}}(L(X))$.

First, we give the definitions to three primitive actions, *Skip*, *Stop* and *Chaos*. Note that we use a lot of properties of $\mathbf{R_t}$, which, however, refer to the laws of $\mathbf{R_{ct}}$, because the similar laws have been proved in the original *Circus Time* [21].

Definition 1.

$$[\![Skip]\!]_{ct} \mathrel{\widehat{=}} \mathbf{R_{ct}}\left(\begin{array}{c}(\neg\ ok \wedge tr \preccurlyeq tr' \wedge front(ref) \leq ref') \vee \\ (ok' \wedge tr' = tr \wedge front(ref') = front(ref) \wedge wait' = wait \wedge state' = state)\end{array}\right)$$

Law 45. $L([\![Skip]\!]_t) = [\![Skip]\!]_{ct}$

Proof.

$$\begin{array}{ll}
& L([\![Skip]\!]_t) \qquad [[\![Skip]\!]_t(2.15)] \\
=& L(\mathbf{R_t}(\exists\, r \bullet r = snd(last(tr_t)) \wedge \mathbb{I}_t)) \qquad [\text{Property 4-L1,L3}] \\
=& \mathbf{R1_{ct}} \circ \mathbf{R3_{ct}}(L(\mathbf{R2_t}(\exists\, r \bullet r = snd(last(tr_t)) \wedge \mathbb{I}_t))) \qquad [\text{predicate calculus}] \\
=& \mathbf{R1_{ct}} \circ \mathbf{R3_{ct}}(L(\mathbf{R2_t}(\mathbb{I}_t))) \qquad [\mathbb{I}_t \text{ and } \mathbf{R2_t}] \\
=& \mathbf{R1_{ct}} \circ \mathbf{R3_{ct}}\left(L\left(\exists\, r \bullet \left(\begin{array}{c}(\neg\ ok \wedge Expands(\langle(\langle\rangle, r)\rangle, dif(tr'_t, tr_t))) \vee \\ (ok' \wedge \langle\langle\rangle, r\rangle = dif(tr'_t, tr_t) \wedge wait' = wait \wedge state' = state)\end{array}\right)\right)\right) \\
& \qquad [Expands(2.2)] \\
=& \mathbf{R1_{ct}} \circ \mathbf{R3_{ct}}\left(L\left(\exists\, r \bullet \left(\begin{array}{c}\left(\begin{array}{c}\neg\ ok \wedge front(\langle(\langle\rangle, r)\rangle) \leq dif(tr'_t, tr_t) \wedge \\ (fst(last(\langle(\langle\rangle, r)\rangle)) \leq fst(dif(tr'_t, tr_t)(\#\langle(\langle\rangle, r)\rangle)))\end{array}\right) \vee \\ (ok' \wedge \langle\langle\rangle, r\rangle = dif(tr'_t, tr_t) \wedge wait' = wait \wedge state' = state)\end{array}\right)\right)\right) \\
& \qquad [\text{properties of sequences}] \\
=& \mathbf{R1_{ct}} \circ \mathbf{R3_{ct}}\left(L\left(\exists\, r \bullet \left(\begin{array}{c}(\ \neg\ ok \wedge \langle\rangle \leq dif(tr'_t, tr_t) \wedge \langle\rangle \leq fst(head(dif(tr'_t, tr_t)))\ ) \vee \\ (ok' \wedge \langle\langle\rangle, r\rangle = dif(tr'_t, tr_t) \wedge wait' = wait \wedge state' = state)\end{array}\right)\right)\right) \\
& \qquad [\text{predicate calculus}] \\
=& \mathbf{R1_{ct}} \circ \mathbf{R3_{ct}}\left(L\left(\begin{array}{c}(\ \neg\ ok \wedge \langle\rangle \leq dif(tr'_t, tr_t) \wedge \langle\rangle \leq fst(head(dif(tr'_t, tr_t)))\ ) \vee \\ \exists\, r \bullet (ok' \wedge \langle\langle\rangle, r\rangle = dif(tr'_t, tr_t) \wedge wait' = wait \wedge state' = state)\end{array}\right)\right) \\
& \qquad [\text{pred. calculus}] \\
=& \mathbf{R1_{ct}} \circ \mathbf{R3_{ct}}\left(L\left(\begin{array}{c}(\ \neg\ ok \wedge \langle\rangle \leq dif(tr'_t, tr_t)\ ) \vee \\ \exists\, r \bullet (ok' \wedge \langle\langle\rangle, r\rangle = dif(tr'_t, tr_t) \wedge wait' = wait \wedge state' = state)\end{array}\right)\right) \\
& \qquad [\text{Property 2-L14}] \\
=& \mathbf{R1_{ct}} \circ \mathbf{R3_{ct}}\left(L\left(\begin{array}{c}(\ \neg\ ok \wedge \langle\rangle \leq proj_fst(dif(tr'_t, tr_t)) \wedge \langle\rangle \leq proj_snd(dif(tr'_t, tr_t))\ ) \vee \\ \exists\, r \bullet (ok' \wedge \langle\langle\rangle, r\rangle = dif(tr'_t, tr_t) \wedge wait' = wait \wedge state' = state)\end{array}\right)\right) \\
& \qquad [\text{Property 2-L15,L16}] \\
=& \mathbf{R1_{ct}} \circ \mathbf{R3_{ct}}\left(L\left(\begin{array}{c}\left(\begin{array}{c}\neg\ ok \wedge \langle\rangle \leq diff(proj_fst(tr'_t), proj_fst(tr_t)) \wedge \\ \langle\rangle \leq proj_snd(tr'_t) - front(proj_snd(tr_t))\end{array}\right) \vee \\ \exists\, r \bullet (ok' \wedge \langle\langle\rangle, r\rangle = dif(tr'_t, tr_t) \wedge wait' = wait \wedge state' = state)\end{array}\right)\right) \\
& \qquad [\text{Property 2-L17}] \\
=& \mathbf{R1_{ct}} \circ \mathbf{R3_{ct}}\left(L\left(\begin{array}{c}\left(\begin{array}{c}\neg\ ok \wedge \langle\rangle \leq diff(proj_fst(tr'_t), proj_fst(tr_t)) \wedge \\ \langle\rangle \leq proj_snd(tr'_t) - front(proj_snd(tr_t))\end{array}\right) \vee \\ \left(\begin{array}{c}(ok' \wedge wait' = wait \wedge state' = state) \wedge \\ \langle\langle\rangle\rangle = diff(proj_fst(tr'_t), proj_fst(tr_t)) \wedge \\ front(proj_snd(tr'_t)) = front(proj_snd(tr_t))\end{array}\right)\end{array}\right)\right) \\
& \qquad [L \text{ and predicate calculus}] \\
=& \mathbf{R1_{ct}} \circ \mathbf{R3_{ct}}\left(\begin{array}{c}(\ \neg\ ok \wedge \langle\rangle \leq diff(tr', tr) \wedge \langle\rangle \leq ref' - front(ref)\ ) \vee \\ \left(\begin{array}{c}(ok' \wedge wait' = wait \wedge state' = state) \wedge \\ \langle\langle\rangle\rangle = diff(tr', tr) \wedge front(ref') = front(ref)\end{array}\right)\end{array}\right) \qquad [\text{properties of sequences}] \\
=& \mathbf{R1_{ct}} \circ \mathbf{R3_{ct}}\left(\begin{array}{c}(\ \neg\ ok \wedge \langle\langle\rangle\rangle \preccurlyeq diff(tr', tr) \wedge front(ref) \leq ref'\ ) \vee \\ \left(\begin{array}{c}(ok' \wedge wait' = wait \wedge state' = state) \wedge \\ \langle\langle\rangle\rangle = diff(tr', tr) \wedge front(ref') = front(ref)\end{array}\right)\end{array}\right) \qquad [\mathbf{R2_{ct}}]
\end{array}$$

$$
\begin{array}{ll}
= \mathbf{R1_{ct}} \circ \mathbf{R3_{ct}} \circ \mathbf{R2_{ct}} \left(\begin{array}{c} \left(\begin{array}{c} \neg\ ok \wedge tr \preccurlyeq tr' \wedge front(ref) \leq ref' \end{array} \right) \vee \\ \left(\begin{array}{c} (ok' \wedge wait' = wait \wedge state' = state) \wedge \\ tr = tr' \wedge front(ref') = front(ref) \end{array} \right) \end{array} \right) & [\mathbf{R_{ct}}] \\
= \mathbf{R_{ct}} \left(\begin{array}{c} (\neg\ ok \wedge tr \preccurlyeq tr' \wedge front(ref) \leq ref') \vee \\ (ok' \wedge tr' = tr \wedge front(ref') = front(ref) \wedge wait' = wait \wedge state' = state) \end{array} \right) & [\ [\![Skip]\!]_{ct}] \\
= [\![Skip]\!]_{ct} &
\end{array}
$$

□

Definition 2.

$$[\![Stop]\!]_{ct} \mathrel{\widehat{=}} \mathbf{CSP1_{ct}}(\mathbf{R3_{ct}}(ok' \wedge wait' \wedge \frown/tr' = \frown/tr))$$

Law 46. $L([\![Stop]\!]_t) = [\![Stop]\!]_{ct}$

Proof.

$$
\begin{array}{ll}
L([\![Stop]\!]_t) & [[\![Stop]\!]_t] \\
= L(\mathbf{CSP1_t}(\mathbf{R3_t}(ok' \wedge wait' \wedge trace' = \langle\rangle))) & [\text{Property 4-L3,L4}] \\
= \mathbf{CSP1_{ct}} \circ \mathbf{R3_{ct}}(L(ok' \wedge wait' \wedge trace' = \langle\rangle)) & [\text{Property 3-L4}] \\
= \mathbf{CSP1_{ct}} \circ \mathbf{R3_{ct}}(ok' \wedge wait' \wedge L(trace' = \langle\rangle)) & [\text{Property 3-L10}] \\
= \mathbf{CSP1_{ct}} \circ \mathbf{R3_{ct}}(ok' \wedge wait' \wedge tr = tr' \wedge \frown/tr' - \frown/tr = \langle\rangle) & [\text{Property of sequences}] \\
= \mathbf{CSP1_{ct}} \circ \mathbf{R3_{ct}}(ok' \wedge wait' \wedge tr = tr' \wedge \frown/tr' = \frown/tr) & [[\![Stop]\!]_{ct}] \\
= [\![Stop]\!]_{ct} &
\end{array}
$$

□

Definition 3.

$$[\![Chaos]\!]_{ct} \mathrel{\widehat{=}} \mathbf{R_{ct}}(\mathbf{true})$$

Law 47. $L([\![Chaos]\!]_t) = [\![Chaos]\!]_{ct}$

Proof.

$$
\begin{array}{ll}
L([\![Chaos]\!]_t) & [[\![Chaos]\!]_t] \\
= L(\mathbf{R_t}(\mathbf{true})) & [\text{Property 4-L1,L3}] \\
= \mathbf{R1_{ct}} \circ \mathbf{R3_{ct}}(L(\mathbf{R2_t}(\mathbf{true}))) & [\text{Law 43}] \\
= \mathbf{R1_{ct}} \circ \mathbf{R3_{ct}}(\mathbf{true}) & [\mathbf{R2_{ct}}] \\
= \mathbf{R1_{ct}} \circ \mathbf{R3_{ct}} \circ \mathbf{R2_{ct}}(\mathbf{true}) & [\ [\![Chaos]\!]_{ct}] \\
= [\![Chaos]\!]_{ct} &
\end{array}
$$

□

Then, we will give the definitions to assignment, prefix and delay, and prove that they equal to the corresponding definitions in the original *Circus Time* model after L.

Definition 4.

$$[\![x := e]\!]_{ct} \mathrel{\widehat{=}} \mathbf{CSP1_{ct}} \left(\mathbf{R_{ct}} \left(\begin{array}{c} ok = ok' \wedge wait = wait' \wedge tr = tr' \wedge \\ front(ref) = front(ref') \wedge state' = state \oplus \{x \mapsto val(e, state)\} \end{array} \right) \right)$$

Law 48. $L([\![x := e]\!]_t) = [\![x := e]\!]_{ct}$

Proof.

$$
\begin{aligned}
& L(\llbracket x := e \rrbracket_t) && [\llbracket x := e \rrbracket_t] \\
=& L(\mathbf{CSP1_t}(\mathbf{R_t}(ok' = ok \wedge wait' = wait \wedge tr'_t = tr_t \wedge state' = state \oplus \{x \mapsto val(e, state)\}))) \\
& && \text{[Property-4-L1,L3,L4]} \\
=& \mathbf{CSP1_{ct}} \circ \mathbf{R1_{ct}} \circ \mathbf{R3_{ct}} \left(L \left(\mathbf{R2_t} \left(\begin{array}{c} ok' = ok \wedge wait' = wait \wedge tr'_t = tr_t \wedge \\ state' = state \oplus \{x \mapsto val(e, state)\} \end{array} \right) \right) \right) && [\mathbf{R2_t}] \\
=& \mathbf{CSP1_{ct}} \circ \mathbf{R1_{ct}} \circ \mathbf{R3_{ct}} \left(L \left(\exists\, r \bullet \left(\begin{array}{c} ok' = ok \wedge wait' = wait \wedge \langle (\langle\rangle, r) \rangle = dif(tr'_t, tr_t) \wedge \\ state' = state \oplus \{x \mapsto val(e, state)\} \end{array} \right) \right) \right) \\
& && \text{[Property-2-L17 and } L\text{]} \\
=& \mathbf{CSP1_{ct}} \circ \mathbf{R1_{ct}} \circ \mathbf{R3_{ct}} \left(\begin{array}{c} ok' = ok \wedge wait' = wait \wedge \langle\langle\rangle\rangle = diff(tr', tr) \wedge \\ front(ref') = front(ref) \wedge state' = state \oplus \{x \mapsto val(e, state)\} \end{array} \right) && [\mathbf{R2_{ct}}] \\
=& \mathbf{CSP1_{ct}} \circ \mathbf{R1_{ct}} \circ \mathbf{R3_{ct}} \circ \mathbf{R2_{ct}} \left(\begin{array}{c} ok' = ok \wedge wait' = wait \wedge tr = tr' \wedge \\ front(ref') = front(ref) \wedge state' = state \oplus \{x \mapsto val(e, state)\} \end{array} \right) \\
& && [\mathbf{R_{ct}}] \\
=& \mathbf{CSP1_{ct}} \circ \mathbf{R_{ct}} \left(\begin{array}{c} ok' = ok \wedge wait' = wait \wedge tr = tr' \wedge \\ front(ref') = front(ref) \wedge state' = state \oplus \{x \mapsto val(e, state)\} \end{array} \right) && [\llbracket x := e \rrbracket_{ct}] \\
=& \llbracket x := e \rrbracket_{ct}
\end{aligned}
$$

□

Definition 5.

$$
\begin{aligned}
& \llbracket c.e \rightarrow Skip \rrbracket_{ct} \mathrel{\widehat{=}} \mathbf{CSP1_{ct}}\,(ok' \wedge \mathbf{R_{ct}}(wait\_com(c)_{ct} \vee terminating\_com(c.e)_{ct})) \\
& wait\_com(c)_{ct} \mathrel{\widehat{=}} \left(\begin{array}{c} wait' \wedge possible(ref, ref', c) \wedge {}^\frown/tr' = {}^\frown/tr \end{array} \right) \\
& possible(ref, ref', c)_{ct} \mathrel{\widehat{=}} \forall\, i : \#ref \,..\, \#ref' \bullet c \notin ref'(i) \\
& term\_com(c.e)_{ct} \mathrel{\widehat{=}} (\neg\, wait' \wedge diff(tr', tr) = \langle\langle c.e \rangle\rangle) \\
& terminating\_com(c.e)_{ct} \mathrel{\widehat{=}} (wait\_com(c)_{ct} \,;\, term\_com(c.e)_{ct})
\end{aligned}
$$

It is also straightforward to prove that $wait\_com(c)_{ct}$ and $term\_com(c.e)_{ct}$ are $\mathbf{R2_{ct}}$ healthy, and the proof is left to the reader for an excise.

Lemma 6. $\mathbf{R2_{ct}}(wait\_com(c)_{ct}) = wait\_com(c)_{ct}$

Lemma 7. $\mathbf{R2_{ct}}(term\_com(c.e)_{ct}) = term\_com(c.e)_{ct}$

Lemma 8. $\mathbf{R2_{ct}}(terminating\_com(c.e)_{ct}) = terminating\_com(c.e)_{ct}$

Law 49. $L(\llbracket c.e \rightarrow Skip \rrbracket_t) = \llbracket c.e \rightarrow Skip \rrbracket_{ct}$

Proof.

$$
\begin{aligned}
& L(\llbracket c.e \rightarrow Skip \rrbracket_t) && [Prefix(2.23) \\
& = L(\mathbf{CSP1_t}(ok' \wedge \mathbf{R_t}(wait\_com(c)_t \vee terminating\_com(c.e)_t))) && \text{[Property 4-L1,L3,L4]} \\
=& \mathbf{CSP1_{ct}}(ok' \wedge \mathbf{R1_{ct}} \circ \mathbf{R3_{ct}}(L(\mathbf{R2_t}(wait\_com(c)_t \vee terminating\_com(c.e)_t)))) && \text{[Lemma 3,5]} \\
=& \mathbf{CSP1_{ct}}(ok' \wedge \mathbf{R1_{ct}} \circ \mathbf{R3_{ct}}(L(wait\_com(c)_t \vee terminating\_com(c.e)_t))) && \text{[Property 3-L11,L13]} \\
=& \mathbf{CSP1_{ct}}(ok' \wedge \mathbf{R1_{ct}} \circ \mathbf{R3_{ct}}(wait\_com(c)_{ct} \vee terminating\_com(c.e)_{ct})) && \text{[Lemma 6,8]} \\
=& \mathbf{CSP1_{ct}}(ok' \wedge \mathbf{R1_{ct}} \circ \mathbf{R3_{ct}}(\mathbf{R2_{ct}}(wait\_com(c)_{ct}) \vee \mathbf{R2_{ct}}(terminating\_com(c.e)_{ct}))) && \text{[Law 10]} \\
=& \mathbf{CSP1_{ct}}(ok' \wedge \mathbf{R_{ct}}(wait\_com(c)_{ct} \vee terminating\_com(c.e)_{ct})) && [Prefix(\text{Definition 5})] \\
=& \llbracket c.e \rightarrow Skip \rrbracket_{ct}
\end{aligned}
$$

□

Definition 6.

$$
\begin{aligned}
& \llbracket Wait\ d \rrbracket_{ct} \mathrel{\widehat{=}} \mathbf{CSP1_{ct}}(\mathbf{R_{ct}}(ok' \wedge {}^\frown/tr = {}^\frown/tr' \wedge delay(d)_{ct})) \\
& delay(d)_{ct} \mathrel{\widehat{=}} \big((\#tr' - \#tr < d) \lhd wait' \rhd \left(\begin{array}{c} \#tr' - \#tr = d \wedge state' = state \end{array} \right)\big)
\end{aligned}
$$

Also, $delay(d)_{ct}$ and $\frown/tr' = \frown/tr$ are $\mathbf{R2_{ct}}$ healthy, and the proof is very straightforward to obtain.

Lemma 9. $\mathbf{R2_{ct}}(delay(d)_{ct}) = delay(d)_{ct}$

Lemma 10. $\mathbf{R2_{ct}}(\frown/tr' = \frown/tr) = \frown/tr' = \frown/tr$

Law 50. $L([\![Wait\ d]\!]_t) = [\![Wait\ d]\!]_{ct}$

Proof.

$$
\begin{array}{ll}
& L(\mathbf{CSP1_t}(\mathbf{R_t}(ok' \wedge delay(d)_t \wedge trace' = \langle\rangle))) \\
& \qquad \text{[Property 4-L1,L3,L4]} \\
= & \mathbf{CSP1_{ct}} \circ \mathbf{R1_{ct}} \circ \mathbf{R3_{ct}}(L(\mathbf{R2_{ct}}((ok' \wedge delay(d)_t \wedge trace' = \langle\rangle)))) \\
& \qquad \text{[Lemma 1,2]} \\
= & \mathbf{CSP1_{ct}} \circ \mathbf{R1_{ct}} \circ \mathbf{R3_{ct}}(L(ok' \wedge delay(d)_t \wedge trace' = \langle\rangle)) \\
& \qquad [L \text{ and } delay(d)_t] \\
= & \mathbf{CSP1_{ct}} \circ \mathbf{R1_{ct}} \circ \mathbf{R3_{ct}} \left(\begin{array}{c} \exists\, tr_t, tr_t', r \bullet \left(\begin{array}{c} tr = proj\_fst(tr_t) \wedge tr' = proj\_fst(tr_t') \wedge \\ ref = proj\_snd(tr_t) \wedge ref' = proj\_snd(tr_t') \end{array} \right) \wedge ok' \wedge \\ ((\#tr_t' - \#tr_t < d) \lhd wait' \rhd (\#tr_t' - \#tr_t = d \wedge state' = state)) \wedge trace' = \langle\rangle \end{array} \right) \\
& \qquad \text{[Property 2-L4]} \\
= & \mathbf{CSP1_{ct}} \circ \mathbf{R1_{ct}} \circ \mathbf{R3_{ct}} \left(\begin{array}{c} \exists\, tr_t, tr_t', r \bullet \left(\begin{array}{c} tr = proj\_fst(tr_t) \wedge tr' = proj\_fst(tr_t') \wedge \\ ref = proj\_snd(tr_t) \wedge ref' = proj\_snd(tr_t') \end{array} \right) \wedge ok' \wedge \\ \left(\begin{array}{c} \#proj\_fst(tr_t') - \#proj\_fst(tr_t) < d \\ \lhd\ wait'\ \rhd \\ (\#proj\_fst(tr_t') - \#proj\_fst(tr_t) = d \wedge state' = state) \end{array} \right) \wedge trace' = \langle\rangle \end{array} \right) \\
& \qquad \text{[Property 3-L10 and predicate calculus]} \\
= & \mathbf{CSP1_{ct}} \circ \mathbf{R1_{ct}} \circ \mathbf{R3_{ct}} \left(\begin{array}{c} ((\#tr' - \#tr < d) \lhd wait' \rhd (\#tr' - \#tr = d \wedge state' = state)) \\ \wedge \frown/tr' - \frown/tr = \langle\rangle \wedge ok' \end{array} \right) \\
& \qquad \text{[Lemma 9,10]} \\
= & \mathbf{CSP1_{ct}} \circ \mathbf{R_{ct}} \left(\begin{array}{c} ((\#tr' - \#tr < d) \lhd wait' \rhd (\#tr' - \#tr = d \wedge state' = state)) \\ \wedge \frown/tr' - \frown/tr = \langle\rangle \wedge ok' \end{array} \right) \quad [[\![Wait\ d]\!]_{ct}] \\
= & [\![Wait\ d]\!]_{ct}
\end{array}
$$

□

Finally, the definitions of external choice, parallel composition and hiding in the original *Circus Time* are transformed into those in the new *Circus Time*.

Definition 7.

$$[\![P \,\Box\, Q]\!]_{ct} \mathrel{\widehat{=}} \mathbf{CSP2_{ct}}((P \wedge Q \wedge Stop) \vee (DifDetected(P, Q)_{ct} \wedge (P \vee Q)))$$

$$DifDetected(P, Q)_{ct} \mathrel{\widehat{=}} \left(\begin{array}{l} \neg\, ok' \vee \\ \left(\begin{array}{l} (ok \wedge \neg\, wait \wedge ((P \wedge Q \wedge ok' \wedge wait' \wedge \frown/tr' = \frown/tr) \vee Skip))\,; \\ \left(\begin{array}{c} (ok' \wedge \neg\, wait' \wedge tr' = tr \wedge ref' = ref \wedge \#tr = \#ref \wedge \#tr' = \#ref') \\ \vee\ (ok' \wedge head(diff(tr', tr)) \neq \langle\rangle) \end{array} \right) \end{array} \right) \end{array} \right)$$

Law 51. $L([\![P \,\Box\, Q]\!]_t) = [\![L(P) \,\Box\, L(Q)]\!]_{ct}$

Proof.

(1)$L([\![P \,\Box\, Q]\!]_t) \Rightarrow [\![L(P) \,\Box\, L(Q)]\!]_{ct}$

$$
\begin{array}{ll}
& L([\![P \,\Box\, Q]\!]_t) \qquad\qquad [[\![P \,\Box\, Q]\!]_t] \\
= & L(\mathbf{CSP2_{ct}}((P \wedge Q \wedge Stop_t) \vee (DifDetected(P,Q)_t \wedge (P \vee Q)))) \qquad \text{[Property 3-L2]} \\
= & L(\mathbf{CSP2_{ct}}(P \wedge Q \wedge Stop_t)) \vee L(DifDetected(P,Q)_t \wedge P) \vee L(DifDetected(P,Q)_{ct} \wedge Q) \\
& \qquad \text{[Property 4-L5]} \\
= & \mathbf{CSP2_{ct}}(L(P \wedge Q \wedge Stop_t)) \vee L(DifDetected(P,Q)_t \wedge P) \vee L(DifDetected(P,Q)_{ct} \wedge Q) \\
& \qquad \text{[Property 3-L3]} \\
\Rightarrow & \mathbf{CSP2_{ct}}(L(P) \wedge L(Q) \wedge L(Stop_t)) \vee (L(DifDetected(P,Q)_t) \wedge (L(P) \vee L(Q))) \\
& \qquad \text{[Property 3-L17 and Law 46]} \\
= & \mathbf{CSP2_{ct}}(L(P) \wedge L(Q) \wedge Stop_{ct}) \vee (DifDetected(P,Q)_{ct} \wedge (L(P) \vee L(Q))) \qquad [[\![L(P) \,\Box\, L(Q)]\!]_{ct}] \\
= & [\![L(P) \,\Box\, L(Q)]\!]_{ct}
\end{array}
$$

(2)$L([\![P \,\Box\, Q]\!]_t) \Leftarrow [\![L(P) \,\Box\, L(Q)]\!]_{ct}$ can be proved in a similar manner.

□

Definition 8.

$$
\begin{array}{l}
[\![P \,|[\, s_1 \mid \{\!|\, CS \,|\!\} \mid s_2 \,]|\, Q]\!]_{ct} \mathrel{\widehat{=}} (P \,;\, U0(out\alpha P) \wedge (Q \,;\, U1(out\alpha Q))_{+\{tr,ref\}} \,;\, M_{\|} \\
M_{\|} \mathrel{\widehat{=}} M_{ct} \,;\, Skip \\
M_{ct} \mathrel{\widehat{=}} \left(\begin{array}{c} ok' = (0.ok \wedge 1.ok) \wedge wait' = (0.wait \vee 1.wait) \wedge \\ MTR(tr, tr', 0.tr, 1.tr, ref, ref', 0.ref, 1.ref) \wedge \\ state' = (0.state - s_2) \oplus (1.state - s_1) \end{array} \right) \\
MTR(tr, tr', 0.tr, 1.tr, ref, ref', 0.ref, 1.ref) \mathrel{\widehat{=}} \\
\left(\begin{array}{c} diff(tr', tr) \in TSync(diff(0.tr, tr), diff(1.tr, tr), CS) \wedge \\ ref' - front(ref) = MRef(0.ref - front(ref), 1.ref - front(ref), CS) \end{array} \right)
\end{array}
$$

Law 52. $L([\![P \,|[\, s_1 \mid \{\!|\, CS \,|\!\} \mid s_2 \,]|\, Q]\!]_t) = [\![L(P) \,|[\, s_1 \mid \{\!|\, CS \,|\!\} \mid s_2 \,]|\, L(Q)]\!]_{ct}$

Proof. (1) $L([\![P \,|[\, s_1 \mid \{\!|\, CS \,|\!\} \mid s_2 \,]|\, Q]\!]_t) \Rightarrow [\![L(P) \,|[\, s_1 \mid \{\!|\, CS \,|\!\} \mid s_2 \,]|\, L(Q)]\!]_{ct}$

$$
\begin{array}{ll}
& L([\![P \,|[\, s_1 \mid \{\!|\, CS \,|\!\} \mid s_2 \,]|\, Q]\!]_t) \\
= & L((P \,;\, U0(out\alpha P) \wedge (Q \,;\, U1(out\alpha Q))_{+\{tr_t\}} \,;\, M_t;\ Skip_t) \qquad \text{[Property 3-L6]} \\
= & L(P \,;\, U0(out\alpha P) \wedge (Q \,;\, U1(out\alpha Q))_{+\{tr_t\}}) \,;\, L(M_t);\ L(Skip_t)) \qquad \text{[Property 3-L15 and Law 45]} \\
= & L(P \,;\, U0(out\alpha P) \wedge (Q \,;\, U1(out\alpha Q))_{+\{tr_t\}}) \,;\, M_{ct};\ Skip_{ct} \\
& \qquad \text{[Property 3-L3,L6 and property of } L \text{ on alphabet extension]} \\
\Rightarrow & ((L(P) \,;\, L(U0(out\alpha P))) \wedge (L(Q) \,;\, L(U1(out\alpha Q)))_{+\{tr,ref\}}) \,;\, M_{ct};\ Skip_{ct} \qquad \text{[Property 3-L18]} \\
= & ((L(P) \,;\, U0(out\alpha L(P))) \wedge (L(Q) \,;\, U1(out\alpha L(Q)))_{+\{tr,ref\}}) \,;\, M_{ct};\ Skip_{ct} \\
= & [\![L(P) \,|[\, s_1 \mid \{\!|\, CS \,|\!\} \mid s_2 \,]|\, L(Q)]\!]_{ct}
\end{array}
$$

(2) $L([\![P \,|[\, s_1 \mid \{\!|\, CS \,|\!\} \mid s_2 \,]|\, Q]\!]_t) \Leftarrow [\![L(P) \,|[\, s_1 \mid \{\!|\, CS \,|\!\} \mid s_2 \,]|\, L(Q)]\!]_{ct}$ can be proved in a similar manner. □

Definition 9.

$$
\begin{array}{l}
P \setminus CS \mathrel{\widehat{=}} \mathbf{R_{ct}}(\exists\, s, r \bullet P[s, r/tr', ref'] \wedge L_{ct}) \,;\, Skip \\
L_{ct} \mathrel{\widehat{=}} diff(tr', tr) = diff(s, tr) \downarrow_{ct} (\Sigma - CS) \wedge r - front(ref) = ((ref' - front(ref)) \cup_{ct} CS) \\
tr_1 = (tr_2 \downarrow_{ct} CS) \Leftrightarrow \forall\, i : 1..\#tr_1 \bullet tr_1(i) = (tr_2(i) \downarrow CS) \wedge \#tr_1 = \#tr_2 \\
ref_1 = (ref_2 \cup_{ct} CS) \Leftrightarrow \forall\, i : 1..\#ref_1 \bullet ref_1(i) = (ref_2(i) \cup CS) \wedge \#ref_1 = \#ref_2
\end{array}
$$

Law 53. $L([\![P \setminus CS]\!]_t) = [\![L(P) \setminus CS]\!]_{ct}$

Proof. The proof of this law can be proved in a similar manner and is left to the reader as an excise. □

There are another four laws about L with conditional choice, guarded action, internal choice and sequential composition. The proofs of these laws are straightforward and left to the reader.

Law 54. $L([\![P \lhd b \rhd Q]\!]_t) = [\![L(P) \lhd b \rhd L(Q)]\!]_{ct}$

Law 55. $L([\![b \& P]\!]_t) = [\![b \& L(P)]\!]_{ct}$

Law 56. $L([\![P \sqcap Q]\!]_t) = [\![L(P) \sqcap L(Q)]\!]_{ct}$

Law 57. $L([\![P;\ Q]\!]_t) = [\![L(P);\ L(Q)]\!]_{ct}$

In short, we give the UTP definitions to all primitive actions and operators of the original *Circus Time* by representing traces and refusals individually, and prove all original operators are $\mathbf{R2_{ct}}$ healthy.

4 Reactive designs

In this section we provide a reactive design semantics that exposes the underlying pre-postcondition semantics that makes it easier to understand the definitions of the operators and even some subtle behaviours. More importantly, the reactive design semantics supports contract-based reasoning about properties of systems. The work in [3, 13] provided the reactive design semantics to some operators in (untimed) CSP. On the other hand, sequential composition, recursion, hiding and so on were not considered. In this report, based on the *Circus Time* model, we develop the reactive design semantics of all operators of our timed theory; we consider a comprehensive number of operators that include all those of Timed CSP, plus operators for deadline specification.

To validate our operator definitions, we calculate them from those of the theory for the original *Circus Time* when available. We start from a transformed definition that adopts the separation of traces and refusals and replaces the healthiness conditions of the original theory by $\mathbf{R_{ct}}$. The calculation commences by applying Theorem 1 to deduce the reactive design semantics for all of the operators. In this section, we give the reactive design to each operator of the *Circus Time* theory with a brief explanation. Nevertheless, we change the definitions of some key operators, such as prefix, external choice and parallelism, for capturing behaviours of actions more precisely. The validation of prefix and external choice is implemented by proving that each of them refines the correspondence of the original theory, and of parallelism is achieved by proving some well-known algebraic laws.

4.1 Primitive Actions

The semantics of *Skip* is given as a program that terminates immediately without changing anything, and its UTP semantics is taken from Definition 1.

$$Skip \mathrel{\widehat{=}} \mathbf{R_{ct}}\left(\begin{array}{c}(\neg\ ok \land tr \preccurlyeq tr' \land front(ref) \leq ref') \lor \\ (ok' \land tr' = tr \land front(ref') = front(ref) \land wait' = wait \land state' = state)\end{array}\right) \tag{4.48}$$

Note that *Skip* in *Circus Time* is different from that in Timed CSP because here time is stopped by the fixed length of traces and the value of $wait'$.

Lemma 11. $Skip^f_f = \mathbf{R1_{ct}}(\neg\ ok)$

Proof.

$$\begin{array}{ll}
& Skip^f_f & [Skip(4.48)] \\
= & \left(\mathbf{R_{ct}}\left(\begin{array}{c}(\neg\ ok \land tr \preccurlyeq tr' \land front(ref) \leq ref') \lor \\ (ok' \land tr' = tr \land front(ref') = front(ref) \land wait' = wait \land state' = state)\end{array}\right)\right)^f_f & [\text{Law 24}] \\
= & \left(\mathbf{R1_{ct}} \circ \mathbf{R2_{ct}}\left(\left(\begin{array}{c}(\neg\ ok \land tr \preccurlyeq tr' \land front(ref) \leq ref') \lor \\ (ok' \land tr' = tr \land front(ref') = front(ref) \land wait' = wait \land state' = state)\end{array}\right)_f\right)\right)^f & \\
& & [\text{substitution for } ok'] \\
= & \mathbf{R1_{ct}} \circ \mathbf{R2_{ct}}(\neg\ ok \land tr \preccurlyeq tr' \land front(ref) \leq ref') & [\text{Law 11}] \\
= & \mathbf{R1_{ct}}(\neg\ ok \land tr \preccurlyeq tr' \land front(ref) \leq ref') & [\mathbf{R1_{ct}} \text{ and prop. calculus}] \\
= & \mathbf{R1_{ct}}(\neg\ ok) &
\end{array}$$

□

Lemma 12. $Skip_f^t = \mathbf{R1_{ct}}(\neg\ ok) \vee \mathbf{R1_{ct}}(\neg\ wait' \wedge tr' = tr \wedge state' = state)$

Proof.

$$
\begin{array}{ll}
& Skip_f^t & [Skip(4.48)] \\
= & \left(\mathbf{R_{ct}}\left(\begin{array}{c}(\neg\ ok \wedge tr \preccurlyeq tr' \wedge front(ref) \leq ref') \vee \\ (ok' \wedge tr' = tr \wedge front(ref') = front(ref) \wedge wait' = wait \wedge state' = state)\end{array}\right)\right)_f^t & [\text{Law 24}] \\
= & \mathbf{R1_{ct}} \circ \mathbf{R2_{ct}}\left(\left(\begin{array}{c}(\neg\ ok \wedge tr \preccurlyeq tr' \wedge front(ref) \leq ref') \vee \\ (ok' \wedge tr' = tr \wedge front(ref') = front(ref) \wedge wait' = wait \wedge state' = state)\end{array}\right)_f\right)^t & \\
& & [\text{substitution for } ok', wait] \\
= & \mathbf{R1_{ct}} \circ \mathbf{R2_{ct}}\left(\begin{array}{c}(\neg\ ok \wedge tr \preccurlyeq tr' \wedge front(ref) \leq ref') \vee \\ (tr' = tr \wedge front(ref') = front(ref) \wedge \neg\ wait' \wedge state' = state)\end{array}\right) & [\text{Law 4,10}] \\
= & \mathbf{R1_{ct}} \circ \mathbf{R2_{ct}}(\neg\ ok \wedge tr \preccurlyeq tr' \wedge front(ref) \leq ref') \vee & [\text{Lemma 11}] \\
& \mathbf{R1_{ct}} \circ \mathbf{R2_{ct}}(tr' = tr \wedge front(ref') = front(ref) \wedge \neg\ wait' \wedge state' = state) & \\
= & \mathbf{R1_{ct}}(\neg\ ok) \vee \mathbf{R1_{ct}} \circ \mathbf{R2_{ct}}(tr' = tr \wedge front(ref') = front(ref) \wedge \neg\ wait' \wedge state' = state) & [\text{Law 11}] \\
= & \mathbf{R1_{ct}}(\neg\ ok) \vee \mathbf{R1_{ct}}(tr' = tr \wedge front(ref') = front(ref) \wedge \neg\ wait' \wedge state' = state) & \\
& & [\mathbf{R1_{ct}} \text{ and property of sequences}] \\
= & \mathbf{R1_{ct}}(\neg\ ok) \vee \mathbf{R1_{ct}}(\neg\ wait' \wedge tr' = tr \wedge state' = state) &
\end{array}
$$

□

Theorem 2. *(Skip)*

$$Skip = \mathbf{R_{ct}}(\mathbf{true} \vdash \neg\ wait' \wedge tr' = tr \wedge state' = state)$$

Proof.

$$
\begin{array}{ll}
& Skip & [\text{Theorem 1}] \\
= & \mathbf{R_{ct}}(\neg\ Skip_f^f \vdash Skip_f^t) & [\text{Lemma 11, 12}] \\
= & \mathbf{R_{ct}}\left(\begin{array}{c}\neg\ \mathbf{R1_{ct}}(\neg\ ok) \vdash \\ \mathbf{R1_{ct}}(\neg\ ok) \vee \mathbf{R1_{ct}}(\neg\ wait' \wedge tr' = tr \wedge state' = state)\end{array}\right) & [\text{design}] \\
= & \mathbf{R_{ct}}\left(\begin{array}{c}\neg\ ok \vee \mathbf{R1_{ct}}(\neg\ ok) \vee (ok' \wedge \mathbf{R1_{ct}}(\neg\ ok)) \vee \\ (ok' \wedge \mathbf{R1_{ct}}(\neg\ wait' \wedge tr' = tr \wedge state' = state))\end{array}\right) & [\text{prop. calc.}] \\
= & \mathbf{R_{ct}}\left(\begin{array}{c}\neg\ ok \vee \mathbf{R1_{ct}}(\neg\ ok) \vee \\ (ok' \wedge \mathbf{R1_{ct}}(\neg\ wait' \wedge tr' = tr \wedge state' = state))\end{array}\right) & [\text{Law 2 and prop. calculus}] \\
= & \mathbf{R_{ct}}(\neg\ ok \vee (ok' \wedge (\neg\ wait' \wedge tr' = tr \wedge state' = state))) & [\text{design}] \\
= & \mathbf{R_{ct}}(\mathbf{true} \vdash \neg\ wait' \wedge tr' = tr \wedge state' = state) &
\end{array}
$$

□

The reactive design semantics of *Skip* is given as an $\mathbf{R_{ct}}$ healthy program in which the precondition is true, and the postcondition states that it terminates immediately without consuming time.

By contrast, the action *Stop* (Definition 2) from the original *Circus Time* is given as a program that waits forever.

$$Stop \mathrel{\widehat{=}} \mathbf{CSP1_{ct}}(\mathbf{R3_{ct}}(ok' \wedge wait' \wedge {}^\frown/tr' = {}^\frown/tr)) \tag{4.49}$$

However, unlike this definition from the original *Circus Time*, we impose $state' = state$ in our new *Circus Time* in order to maintain consistency with other operators. Therefore, the UTP definition of *Stop* is given as

$$Stop \mathrel{\widehat{=}} \mathbf{CSP1_{ct}}(\mathbf{R3_{ct}}(ok' \wedge wait' \wedge {}^\frown/tr' = {}^\frown/tr \wedge state' = state)) \tag{4.50}$$

And its reactive design semantics is deduced as follows. As usual, the precondition and postcondition are calculated in Lemma 13 (whose proof is left to the reader) and Lemma 14.

Lemma 13. $Stop_f^f = \mathbf{R1_{ct}}(\neg\, ok)$

Lemma 14. $Stop_f^t = \mathbf{R1_{ct}}(\neg\, ok) \vee (wait' \wedge \frown/tr' = \frown/tr \wedge state' = state)$

Proof.

$$
\begin{aligned}
& Stop_f^t && [Stop]\\
=&(\mathbf{CSP1_{ct}}(\mathbf{R3_{ct}}(ok' \wedge wait' \wedge \frown/tr' = \frown/tr \wedge state' = state)))_f^t && [\mathbf{CSP1_{ct}}]\\
=&((\neg\, ok \wedge RT) \vee (ok' \wedge (\mathbf{R3_{ct}}(ok' \wedge wait' \wedge \frown/tr' = \frown/tr \wedge state' = state))))_f^t && [\text{subs.}]\\
=&((\neg\, ok \wedge RT) \vee (\mathbf{true} \wedge (\mathbf{R3_{ct}}(true \wedge wait' \wedge \frown/tr' = \frown/tr \wedge state' = state))_f)) \\
& && [\text{substitution and } \neg\, wait \text{ in } \mathbf{R3_{ct}}]\\
=&(\neg\, ok \wedge RT) \vee (wait' \wedge \frown/tr' = \frown/tr \wedge state' = state) && [\mathbf{R1_{ct}}]\\
=&\mathbf{R1_{ct}}(\neg\, ok) \vee (wait' \wedge \frown/tr' = \frown/tr \wedge state' = state)
\end{aligned}
$$

□

Theorem 3. *(Stop)*

$$Stop \mathrel{\widehat{=}} \mathbf{R_{ct}}(\mathbf{true} \vdash wait' \wedge \frown/tr' = \frown/tr)$$

Proof.

$$
\begin{aligned}
& Stop && [\text{Theorem 1}]\\
=&\mathbf{R_{ct}}(\neg\, Stop_f^f \vdash Stop_f^t) && [\text{Lemma 13, 14}]\\
=&\mathbf{R_{ct}}(\neg\, \mathbf{R1_{ct}}(\neg\, ok) \vdash \mathbf{R1_{ct}}(\neg\, ok) \vee (wait' \wedge \frown/tr' = \frown/tr \wedge state' = state)) && [\text{design}]\\
=&\mathbf{R_{ct}}(\neg\, ok \vee \mathbf{R1_{ct}}(\neg\, ok) \vee (ok' \wedge \mathbf{R1_{ct}}(\neg\, ok)) \vee (ok' \wedge (wait' \wedge \frown/tr' = \frown/tr \wedge state' = state))) \\
& && [\text{prop. calculus}]\\
=&\mathbf{R_{ct}}(\neg\, ok \vee (ok' \wedge (wait' \wedge \frown/tr' = \frown/tr \wedge state' = state))) && [\text{design}]\\
=&\mathbf{R_{ct}}(\mathbf{true} \vdash wait' \wedge \frown/tr' = \frown/tr \wedge state' = state)
\end{aligned}
$$

□

This reactive design states that *Stop* is a deadlocked program in which, in the postcondition, the trace is extending to denote the lapse of time but not execute any observable events ($\frown/tr' = \frown/tr$), and moreover any refusal set is a valid observation since ref' is unconstrained.

The actions *Chaos* and *Miracle* are given as the $\mathbf{R_{ct}}$-healthy design *Chaos* and *Miracle* respectively.

Theorem 4. $Chaos \mathrel{\widehat{=}} \mathbf{R_{ct}}(\mathbf{false} \vdash \mathbf{true})$

Theorem 5. $Miracle = \mathbf{R_{ct}}(\mathbf{true} \vdash \mathbf{false})$

The above two reactive designs state that if the precondition is **false**, the predicate *Chaos* is **true** no matter what the postcondition is, and if the precondition is *true*, the predicate *Miracle* is miraculous whereas its postcondition is **false**. We can further explore the behaviour of *Miracle* by unfolding the definitions of the healthiness conditions.

$$
\begin{aligned}
& Miracle && [Miracle]\\
=&\mathbf{R_{ct}}(\mathbf{true} \vdash \mathbf{false}) && [\text{design}]\\
=&\mathbf{R_{ct}}(\mathbf{true} \wedge ok \Rightarrow ok' \wedge \mathbf{false}) && [\text{propositional calculus}]\\
=&\mathbf{R_{ct}}(\neg\, ok) && [\mathbf{R3_{ct}}]\\
=&\mathbf{R1_{ct}} \circ \mathbf{R2_{ct}}(\mathbb{I}_{ct} \lhd wait \rhd \neg\, ok) && [\text{relational calculus}]\\
=&\mathbf{R1_{ct}} \circ \mathbf{R2_{ct}}((\mathbb{I}_{ct} \wedge wait) \vee (\neg\, ok \wedge \neg\, wait)) && [\text{Law 4,10}]\\
=&\mathbf{R1_{ct}} \circ \mathbf{R2_{ct}}(\mathbb{I}_{ct} \wedge wait) \vee \mathbf{R1_{ct}} \circ \mathbf{R2_{ct}}(\neg\, ok \wedge \neg\, wait) && [\text{Law 11 and predicate calculus}]\\
=&\mathbf{R1_{ct}}(\mathbb{I}_{ct} \wedge wait) \vee \mathbf{R1_{ct}}(\neg\, ok \wedge \neg\, wait) && [\text{Law 5}]\\
=&(\mathbb{I}_{ct} \wedge wait) \vee \mathbf{R1_{ct}}(\neg\, ok \wedge \neg\, wait) && [\mathbb{I}_{ct} \text{ and } \mathbf{R1_{ct}}]\\
=&(\neg\, ok \wedge RT \wedge wait) \vee (ok' \wedge \mathbb{I} \wedge wait) \vee (\neg\, ok \wedge \neg\, wait \wedge RT) && [\text{propositional calculus and } \mathbf{R1_{ct}}]\\
=&\mathbf{R1_{ct}}(\neg\, ok) \vee (ok' \wedge \mathbb{I} \wedge wait)
\end{aligned}
$$

The observations of *Miracle* are described in two parts: the left disjunct states that its predecessor diverges and *Miracle* is in an unstable state (since ok is false), and the second one states that *Miracle* is waiting for its predecessor's termination (i.e. *wait* is true) but in a stable state (i.e., ok' is true). In both cases, this action has not started yet. As an unstarted action, *Miracle* must not be seen during the execution of actions.

4.2 Sequential Composition

The definition of sequential composition in the original *Circus Time* is the same as that in the UTP.

$$P\,;\,Q \mathrel{\widehat=} \exists\, obs_0 \bullet P[obs_0/obs'] \wedge Q[obs_0/obs] \;\; \textit{if}\;\; out\alpha P = in\alpha Q' = \{obs'\}$$
$$in\alpha(P\,;\,Q) = in\alpha P \qquad out\alpha(P\,;\,Q) = out\alpha Q$$

To deduce the reactive design of sequential composition, we first give an auxiliary law.

Law 58. *Suppose P is* $\mathbf{R1_{ct}}$ *and* $\mathbf{CSP1_{ct}}$, $\mathbf{R1_{ct}}(\neg\, ok)\,;\,P = \mathbf{R1_{ct}}(\neg\, ok)$

Proof.

$$\begin{array}{rl}
& \mathbf{R1_{ct}}(\neg\, ok)\,;\,P & [\mathbf{R1_{ct}}] \\
=& (\neg\, ok \wedge RT)\,;\,P & \text{[relational calculus]} \\
=& \neg\, ok \wedge (RT\,;\,P) & \text{[Lemma 42]} \\
=& \neg\, ok \wedge RT & [\mathbf{R1_{ct}}] \\
=& \mathbf{R1_{ct}}(\neg\, ok)
\end{array}$$

□

Below, we characterise the behaviour of the sequential composition of two $\mathbf{R1_{ct}}$-healthy predicates.

Law 59. *Suppose P, Q, R and S are predicates (ok,ok' are not in αP, αQ, αR and αS),*

$$\mathbf{R1_{ct}}(P \vdash Q)\,;\,\mathbf{R1_{ct}}(R \vdash S) = \mathbf{R1_{ct}}\left(\begin{array}{c} \neg\,(\mathbf{R1_{ct}}(\neg\, P)\,;\,\mathbf{R1_{ct}}(\mathbf{true})) \wedge \neg\,(\mathbf{R1_{ct}}(Q)\,;\,\mathbf{R1_{ct}}(\neg\, R)) \\ \vdash \\ \mathbf{R1_{ct}}(Q)\,;\,\mathbf{R1_{ct}}(S) \end{array}\right)$$

Proof.

$$
\begin{aligned}
&\mathbf{R1_{ct}}(P \vdash Q)\,;\,\mathbf{R1_{ct}}(R \vdash S) && \text{[def of design]}\\
=&\mathbf{R1_{ct}}(\neg\, ok \vee \neg\, P \vee (ok' \wedge Q))\,;\,\mathbf{R1_{ct}}(\neg\, ok \vee \neg\, R \vee (ok' \wedge S)) && \text{[Law 4 and rel. cal.]}\\
=&\mathbf{R1_{ct}}(\neg\, ok)\,;\,\mathbf{R1_{ct}}(\neg\, ok \vee \neg\, R \vee (ok' \wedge S)) \vee \mathbf{R1_{ct}}(\neg\, P)\,;\,\mathbf{R1_{ct}}(\neg\, ok \vee \neg\, R \vee (ok' \wedge S)) \vee\\
&\mathbf{R1_{ct}}(ok' \wedge Q)\,;\,\mathbf{R1_{ct}}(\neg\, ok \vee \neg\, R \vee (ok' \wedge S)) && \text{[Law 58]}\\
=&\mathbf{R1_{ct}}(\neg\, ok) \vee \mathbf{R1_{ct}}(\neg\, P)\,;\,\mathbf{R1_{ct}}(\neg\, ok \vee \neg\, R \vee (ok' \wedge S)) \vee && \text{[def of ; and case split on } ok\text{]}\\
&\mathbf{R1_{ct}}(ok' \wedge Q)\,;\,\mathbf{R1_{ct}}(\neg\, ok \vee \neg\, R \vee (ok' \wedge S))\\
=&\mathbf{R1_{ct}}(\neg\, ok) \vee \mathbf{R1_{ct}}(\neg\, P)[\mathbf{false}/ok']\,;\,\mathbf{R1_{ct}}(\neg\, ok \vee \neg\, R \vee (ok' \wedge S))[\mathbf{false}/ok] \vee && \text{[substitution]}\\
&\mathbf{R1_{ct}}(\neg\, P)[\mathbf{true}/ok']\,;\,\mathbf{R1_{ct}}(\neg\, ok \vee \neg\, R \vee (ok' \wedge S))[\mathbf{true}/ok] \vee\\
&\mathbf{R1_{ct}}(ok' \wedge Q)\,;\,\mathbf{R1_{ct}}(\neg\, ok \vee \neg\, R \vee (ok' \wedge S))\\
=&\mathbf{R1_{ct}}(\neg\, ok) \vee (\mathbf{R1_{ct}}(\neg\, P)\,;\,\mathbf{R1_{ct}}(\mathbf{true})) \vee && \text{[relational calculus]}\\
&\mathbf{R1_{ct}}(\neg\, P)\,;\,\mathbf{R1_{ct}}(\neg\, R \vee (ok' \wedge S)) \vee \mathbf{R1_{ct}}(ok' \wedge Q)\,;\,\mathbf{R1_{ct}}(\neg\, ok \vee \neg\, R \vee (ok' \wedge S))\\
=&\mathbf{R1_{ct}}(\neg\, ok) \vee \mathbf{R1_{ct}}(\neg\, P)\,;\,(\mathbf{R1_{ct}}(\mathbf{true}) \vee \mathbf{R1_{ct}}(\neg\, R \vee (ok' \wedge S))) \vee && \text{[Law 4 and prop. calculus]}\\
&\mathbf{R1_{ct}}(ok' \wedge Q)\,;\,\mathbf{R1_{ct}}(\neg\, ok \vee \neg\, R \vee (ok' \wedge S))\\
=&\mathbf{R1_{ct}}(\neg\, ok) \vee (\mathbf{R1_{ct}}(\neg\, P)\,;\,\mathbf{R1_{ct}}(\mathbf{true})) \vee \mathbf{R1_{ct}}(ok' \wedge Q)\,;\,\mathbf{R1_{ct}}(\neg\, ok \vee \neg\, R \vee (ok' \wedge S)) && \text{[prop. cal.]}\\
=&\mathbf{R1_{ct}}(\neg\, ok) \vee (\mathbf{R1_{ct}}(\neg\, P);\,\mathbf{R1_{ct}}(\mathbf{true})) \vee && \text{[relational calculus and Law 4]}\\
&\mathbf{R1_{ct}}(ok' \wedge Q)\,;\,\mathbf{R1_{ct}}(\neg\, ok \vee (ok \wedge \neg\, R) \vee (ok \wedge ok' \wedge S))\\
=&\mathbf{R1_{ct}}(\neg\, ok) \vee (\mathbf{R1_{ct}}(\neg\, P)\,;\,\mathbf{R1_{ct}}(\mathbf{true})) \vee && \text{[relational calculus]}\\
&(\mathbf{R1_{ct}}(ok' \wedge Q)\,;\,\mathbf{R1_{ct}}(\neg\, ok)) \vee (\mathbf{R1_{ct}}(ok' \wedge Q)\,;\,\mathbf{R1_{ct}}(ok \wedge \neg\, R)) \vee\\
&(\mathbf{R1_{ct}}(ok' \wedge Q)\,;\,\mathbf{R1_{ct}}(ok \wedge ok' \wedge S))\\
=&\mathbf{R1_{ct}}(\neg\, ok) \vee (\mathbf{R1_{ct}}(\neg\, P)\,;\,\mathbf{R1_{ct}}(\mathbf{true})) \vee && \text{[propositional calculus]}\\
&\mathbf{false} \vee (\mathbf{R1_{ct}}(Q)\,;\,\mathbf{R1_{ct}}(\neg\, R)) \vee (\mathbf{R1_{ct}}(Q)\,;\,\mathbf{R1_{ct}}(ok' \wedge S))\\
=&\mathbf{R1_{ct}}(\neg\, ok) \vee (\mathbf{R1_{ct}}(\neg\, P)\,;\,\mathbf{R1_{ct}}(\mathbf{true})) \vee && \text{[def of design]}\\
&(\mathbf{R1_{ct}}(Q)\,;\,\mathbf{R1_{ct}}(\neg\, R)) \vee ((\mathbf{R1_{ct}}(Q)\,;\,\mathbf{R1_{ct}}(S)) \wedge ok')\\
=&\mathbf{R1_{ct}}\left(\ \neg\,(\mathbf{R1_{ct}}(\neg\, P)\,;\,\mathbf{R1_{ct}}(\mathbf{true})) \wedge \neg\,(\mathbf{R1_{ct}}(Q)\,;\,\mathbf{R1_{ct}}(\neg\, R)) \vdash \mathbf{R1_{ct}}(Q)\,;\,\mathbf{R1_{ct}}(S)\ \right)
\end{aligned}
$$

□

Here Law 59 states that the sequential composition of two $\mathbf{R1_{ct}}$-healthy designs is still a $\mathbf{R1_{ct}}$-healthy design. $\mathbf{R2_{ct}}$ has the same property of designs through the composition.

Law 60. *($\mathbf{R2_{ct}}$-design)*

$$\mathbf{R2_{ct}}(P \vdash Q) = (\neg\, \mathbf{R2_{ct}}(\neg\, P)) \vdash \mathbf{R2_{ct}}(Q)$$

Proof.

$$
\begin{aligned}
&\mathbf{R2_{ct}}(P \vdash Q) && \text{[design]}\\
=&\mathbf{R2_{ct}}(\neg\, ok \vee \neg\, P \vee (ok' \wedge Q)) && \text{[Law 10 and predicate calculus]}\\
=&\neg\, ok \vee \mathbf{R2_{ct}}(\neg\, P) \vee (ok' \wedge \mathbf{R2_{ct}}(Q)) && \text{[design]}\\
=&(\neg\, \mathbf{R2_{ct}}(\neg\, P)) \vdash \mathbf{R2_{ct}}(Q)
\end{aligned}
$$

□

However, $\mathbf{R3_{ct}}$ does not preserve the similar property because the *Circus Time* identity II_{ct} is not a design, and hence $\mathbf{R3_{ct}}(P)$ is not, even if P is a design. For this reason, Woodcock in [24] introduces a new healthiness condition to replace **R3**, in order to make a design behave like the identity design when waiting. And here we have a similar healthiness condition, $\mathbf{R3j_{ct}}(P) \mathrel{\widehat{=}} \mathit{II}_D \lhd \mathit{wait} \rhd P$ where $\mathit{II}_D = \mathbf{true} \vdash \mathit{II}$. We also have a useful law about the new healthiness condition.

Law 61. $\mathbf{R1_{ct}} \circ \mathbf{R3j_{ct}} = \mathbf{R1_{ct}} \circ \mathbf{R3_{ct}}$

Proof.

$$
\begin{aligned}
& \mathbf{R1_{ct}} \circ \mathbf{R3j_{ct}}(P) && [] \\
=& \mathbf{R1_{ct}}(\mathit{II}_D \lhd \mathit{wait} \rhd P) && [\text{Law 7}] \\
=& \mathbf{R1_{ct}}(\mathbf{R1_{ct}}(\mathit{II}_D) \lhd \mathit{wait} \rhd P) && [\mathit{II}_D] \\
=& \mathbf{R1_{ct}}(\mathbf{R1_{ct}}(\neg\, \mathit{ok} \vee (\mathit{ok}' \wedge \mathit{II})) \lhd \mathit{wait} \rhd P) && [\text{Law 4}] \\
=& \mathbf{R1_{ct}}(((\neg\, \mathit{ok} \wedge \mathit{RT}) \vee (\mathit{ok}' \wedge \mathit{II})) \lhd \mathit{wait} \rhd P) && [\mathit{II}_t] \\
=& \mathbf{R1_{ct}}(\mathit{II}_t \lhd \mathit{wait} \rhd P) && [\mathbf{R3_{ct}}] \\
=& \mathbf{R1_{ct}} \circ \mathbf{R3_{ct}}(P)
\end{aligned}
$$

□

Finally, we are ready to deduce the reactive design of sequential composition from its original definition.

Theorem 6. *(***Sequential composition***) Suppose P and Q are two Circus Time actions,*

$$
P\,;\, Q = \mathbf{R_{ct}} \left(\begin{array}{c} \neg\, (\mathbf{R1_{ct}}(P_f^f)\,;\, \mathbf{R1_{ct}}(\mathbf{true})) \wedge \neg\, (\mathbf{R1_{ct}}(P_f^t)\,;\, \mathbf{R1_{ct}}(\neg\, \mathit{wait} \wedge \mathbf{R2_{ct}}(Q_f^f))) \\ \vdash \\ \mathbf{R1_{ct}}(P_f^t)\,;\, \mathbf{R1_{ct}}(\mathit{II} \lhd \mathit{wait} \rhd \mathbf{R2_{ct}}(Q_f^t)) \end{array} \right)
$$

Proof.

$$
\begin{aligned}
& P\,;\, Q && [\text{Theorem 1}] \\
=\ & \mathbf{R_{ct}}(\neg\, P_f^f \vdash P_f^t)\,;\, \mathbf{R_{ct}}(\neg\, Q_f^f \vdash Q_f^t) && [\mathbf{R_{ct}}] \\
=\ & \mathbf{R2_{ct}} \circ \mathbf{R3_{ct}} \circ\ \mathbf{R1_{ct}}(\neg\, P_f^f \vdash P_f^t)\,;\, \mathbf{R2_{ct}} \circ \mathbf{R1_{ct}} \circ \mathbf{R3_{ct}}(\neg\, Q_f^f \vdash Q_f^t) && [\text{Law 12}] \\
=\ & \mathbf{R2_{ct}} \circ (\mathbf{R3_{ct}} \circ \mathbf{R1_{ct}}(\neg\, P_f^f \vdash P_f^t)\,;\, \mathbf{R2_{ct}} \circ \mathbf{R1_{ct}} \circ \mathbf{R3_{ct}}(\neg\, Q_f^f \vdash Q_f^t)) && [\mathbf{R1_{ct}}, \mathbf{R2_{ct}}, \mathbf{R3_{ct}} \text{ are commutative}] \\
=\ & \mathbf{R2_{ct}} \circ (\mathbf{R3_{ct}} \circ \mathbf{R1_{ct}}(\neg\, P_f^f \vdash P_f^t)\,;\, \mathbf{R1_{ct}} \circ \mathbf{R3_{ct}} \circ \mathbf{R2_{ct}}(\neg\, Q_f^f \vdash Q_f^t)) && [\text{Law 15}] \\
=\ & \mathbf{R2_{ct}} \circ \mathbf{R3_{ct}} \circ (\mathbf{R1_{ct}}(\neg\, P_f^f \vdash P_f^t)\,;\, \mathbf{R1_{ct}} \circ \mathbf{R3_{ct}} \circ \mathbf{R2_{ct}}(\neg\, Q_f^f \vdash Q_f^t)) && [\text{Law 61}] \\
=\ & \mathbf{R2_{ct}} \circ \mathbf{R3_{ct}} \circ (\mathbf{R1_{ct}}(\neg\, P_f^f \vdash P_f^t);\, \mathbf{R1_{ct}} \circ \mathbf{R3j_{ct}} \circ \mathbf{R2_{ct}}(\neg\, Q_f^f \vdash Q_f^t)) && [\text{Law 60}] \\
=\ & \mathbf{R2_{ct}} \circ \mathbf{R3_{ct}} \circ (\mathbf{R1_{ct}}(\neg\, P_f^f \vdash P_f^t)\,;\, \mathbf{R1_{ct}} \circ \mathbf{R3j_{ct}}(\neg\, \mathbf{R2_{ct}}(Q_f^f) \vdash \mathbf{R2_{ct}}(Q_f^t))) && [\mathbf{R3j_{ct}}] \\
=\ & \mathbf{R2_{ct}} \circ \mathbf{R3_{ct}} \circ (\mathbf{R1_{ct}}(\neg\, P_f^f \vdash P_f^t)\,;\, \mathbf{R1_{ct}}(\mathit{II}_D \lhd \mathit{wait} \rhd \neg\, \mathbf{R2_{ct}}(Q_f^f) \vdash \mathbf{R2_{ct}}(Q_f^t))) && [\mathit{II}_D] \\
=\ & \mathbf{R2_{ct}} \circ \mathbf{R3_{ct}}(\mathbf{R1_{ct}}(\neg\, P_f^f \vdash P_f^t)\,;\, \mathbf{R1_{ct}}(\mathbf{true} \vdash \mathit{II} \lhd \mathit{wait} \rhd \neg\, \mathbf{R2_{ct}}(Q_f^f) \vdash \mathbf{R2_{ct}}(Q_f^t))) && [\text{Property 1-L2}]
\end{aligned}
$$

$$
= \mathbf{R2_{ct}} \circ \mathbf{R3_{ct}}(\mathbf{R1_{ct}}(\neg\, P_f^f \vdash P_f^t)\,;\, \mathbf{R1_{ct}}(\mathbf{true} \lhd \mathit{wait} \rhd \neg\, \mathbf{R2_{ct}}(Q_f^f) \vdash \mathit{II} \lhd \mathit{wait} \rhd \mathbf{R2_{ct}}(Q_f^t)))
$$

[conditional choice and propositional calculus]

$$
\begin{aligned}
=\ & \mathbf{R2_{ct}} \circ \mathbf{R3_{ct}}(\mathbf{R1_{ct}}(\neg\, P_f^f \vdash P_f^t)\,;\, \mathbf{R1_{ct}}(\mathit{wait} \vee \neg\, \mathbf{R2_{ct}}(Q_f^f) \vdash \mathit{II} \lhd \mathit{wait} \rhd \mathbf{R2_{ct}}(Q_f^t))) && [\text{Law 59}] \\
=\ & \mathbf{R2_{ct}} \circ \mathbf{R3_{ct}} \circ \mathbf{R1_{ct}} \left(\begin{array}{c} \neg\, (\mathbf{R1_{ct}}(P_f^f)\,;\, \mathbf{R1_{ct}}(\mathbf{true})) \wedge \neg\, (\mathbf{R1_{ct}}(P_f^t)\,;\, \mathbf{R1_{ct}}(\neg\, \mathit{wait} \wedge \mathbf{R2_{ct}}(Q_f^f))) \\ \vdash \\ \mathbf{R1_{ct}}(P_f^t)\,;\, \mathbf{R1_{ct}}(\mathit{II} \lhd \mathit{wait} \rhd \mathbf{R2_{ct}}(Q_f^t)) \end{array} \right) && [\mathbf{R_{ct}}] \\
=\ & \mathbf{R_{ct}} \left(\begin{array}{c} \neg\, (\mathbf{R1_{ct}}(P_f^f)\,;\, \mathbf{R1_{ct}}(\mathbf{true})) \wedge \neg\, (\mathbf{R1_{ct}}(P_f^t)\,;\, \mathbf{R1_{ct}}(\neg\, \mathit{wait} \wedge \mathbf{R2_{ct}}(Q_f^f))) \\ \vdash \\ \mathbf{R1_{ct}}(P_f^t)\,;\, \mathbf{R1_{ct}}(\mathit{II} \lhd \mathit{wait} \rhd \mathbf{R2_{ct}}(Q_f^t)) \end{array} \right)
\end{aligned}
$$

□

This theorem shows that, if P does not diverge and Q does not diverge after P terminates, $P\,;\, Q$ behaves like the composition of the terminations of P and subsequent Q.

4.3 Assignment

The assignment operator assigns a value to a local variable and takes no time. The definition (Definition 4) in the original *Circus Time* is given as

$$x := e \mathrel{\widehat{=}} \mathbf{CSP1_{ct}}\left(\mathbf{R_{ct}}\left(\begin{array}{c} ok = ok' \wedge wait = wait' \wedge tr = tr' \wedge front(ref) = front(ref') \wedge \\ state' = state \oplus \{x \mapsto val(e, state)\} \end{array}\right)\right)$$

where $\oplus$ is an override operator and an evaluation function $val(e, state)$ returns the value of the expression e in *state*.

The reactive design of the assignment operator is calculated as follows.

Lemma 15. $\mathbf{R2_{ct}}(tr' = tr) = (tr' = tr)$

Lemma 16. $(x := e)^f_f = \mathbf{R1_{ct}}(\neg\ ok)$

Proof.

$$\begin{array}{ll}
& (x := e)^f_f \qquad \text{[Definition 4]} \\
= & \left(\mathbf{CSP1_{ct}}\left(\mathbf{R_{ct}}\left(\begin{array}{c} ok = ok' \wedge wait = wait' \wedge tr = tr' \wedge front(ref) = front(ref') \wedge \\ state' = state \oplus \{x \mapsto val(e, state)\} \end{array}\right)\right)\right)^f_f \qquad [\mathbf{CSP1_{ct}}] \\
= & \left((\neg\ ok \wedge RT) \vee \left(\mathbf{R_{ct}}\left(\begin{array}{c} ok = ok' \wedge wait = wait' \wedge tr = tr' \wedge front(ref) = front(ref') \wedge \\ state' = state \oplus \{x \mapsto val(e, state)\} \end{array}\right)\right)\right)^f_f \\
& \qquad \text{[substitution for } wait \text{ and } \mathbf{R3_{ct}}] \\
= & \left((\neg\ ok \wedge RT) \vee \left(\mathbf{R1_{ct}} \circ \mathbf{R2_{ct}}\left(\begin{array}{c} ok = ok' \wedge \neg\ wait' \wedge tr = tr' \wedge front(ref) = front(ref') \wedge \\ state' = state \oplus \{x \mapsto val(e, state)\} \end{array}\right)\right)\right)^f \\
& \qquad \text{[substitution for } ok'] \\
= & (\neg\ ok \wedge RT) \vee \left(\mathbf{R1_{ct}} \circ \mathbf{R2_{ct}}\left(\begin{array}{c} \neg\ ok \wedge \neg\ wait' \wedge tr = tr' \wedge front(ref) = front(ref') \wedge \\ state' = state \oplus \{x \mapsto val(e, state)\} \end{array}\right)\right) \\
& \qquad \text{[Lemma 15 and } \mathbf{R1_{ct}}] \\
= & (\neg\ ok \wedge RT) \vee \left(\begin{array}{c} (\neg\ ok \wedge RT) \wedge \neg\ wait' \wedge tr = tr' \wedge \\ front(ref) = front(ref') \wedge state' = state \oplus \{x \mapsto val(e, state)\} \end{array}\right) \\
& \qquad \text{[propositional calculus]} \\
= & \neg\ ok \wedge RT \qquad [\mathbf{R1_{ct}}] \\
= & \mathbf{R1_{ct}}(\neg\ ok)
\end{array}$$

□

Lemma 17. $(x := e)^t_f = \mathbf{R1_{ct}}(\neg\ ok) \vee \mathbf{R1_{ct}}(\neg\ wait' \wedge tr = tr' \wedge state' = state \oplus \{x \mapsto val(e, state)\})$

Proof.

$$\begin{array}{ll}
& (x := e)^t_f \qquad \text{[Definition 4]} \\
= & \left(\mathbf{CSP1_{ct}}\left(\mathbf{R_{ct}}\left(\begin{array}{c} ok = ok' \wedge wait = wait' \wedge tr = tr' \wedge front(ref) = front(ref') \wedge \\ state' = state \oplus \{x \mapsto val(e, state)\} \end{array}\right)\right)\right)^t_f \qquad [\mathbf{CSP1_{ct}}] \\
= & \left((\neg\ ok \wedge RT) \vee \left(\mathbf{R_{ct}}\left(\begin{array}{c} ok = ok' \wedge wait = wait' \wedge tr = tr' \wedge front(ref) = front(ref') \wedge \\ state' = state \oplus \{x \mapsto val(e, state)\} \end{array}\right)\right)\right)^t_f \\
& \qquad \text{[substitution for } wait, ok' \text{ and } \mathbf{R3_{ct}}] \\
= & (\neg\ ok \wedge RT) \vee \left(\mathbf{R1_{ct}} \circ \mathbf{R2_{ct}}\left(\begin{array}{c} ok \wedge \neg\ wait' \wedge tr = tr' \wedge front(ref) = front(ref') \wedge \\ state' = state \oplus \{x \mapsto val(e, state)\} \end{array}\right)\right) \\
& \qquad \text{[Lemma 15 and } \mathbf{R1_{ct}}] \\
= & (\neg\ ok \wedge RT) \vee (\mathbf{R1_{ct}}(ok \wedge \neg\ wait' \wedge tr = tr' \wedge state' = state \oplus \{x \mapsto val(e, state)\})) \\
& \qquad [\mathbf{R1_{ct}} \text{ and propositional calculus}] \\
= & \mathbf{R1_{ct}}(\neg\ ok) \vee \mathbf{R1_{ct}}(\neg\ wait' \wedge tr = tr' \wedge state' = state \oplus \{x \mapsto val(e, state)\})
\end{array}$$

□

Theorem 7.

$$x := e \mathrel{\widehat{=}} \mathbf{R_{ct}}\left(\mathbf{true} \vdash \left(\ \neg\ wait' \wedge tr' = tr \wedge state' = state \oplus \{x \mapsto val(e, state)\}\ \right)\right)$$

Proof.

$$\begin{aligned}
& x := e && \text{[Theorem 1}\\
=& \mathbf{R_{ct}}(\neg\ (x := e)^f_f \vdash (x := e)^t_f) && \text{[design]}\\
=& \mathbf{R_{ct}}(\neg\ ok \vee (x := e)^f_f \vee (ok' \wedge (x := e)^t_f)) && \text{[Lemma 16,17]}\\
=& \mathbf{R_{ct}}\left(\begin{array}{c}\neg\ ok \vee \mathbf{R1_{ct}}(\neg\ ok) \vee \\ (ok' \wedge \mathbf{R1_{ct}}(\neg\ ok) \vee \mathbf{R1_{ct}}(\neg\ wait' \wedge tr = tr' \wedge state' = state \oplus \{x \mapsto val(e, state)\}))\end{array}\right) && \text{[Law 2]}\\
=& \mathbf{R_{ct}}(\neg\ ok \vee (ok' \wedge (\neg\ wait' \wedge tr = tr' \wedge state' = state \oplus \{x \mapsto val(e, state)\}))) && \text{[design]}\\
=& \mathbf{R_{ct}}(\mathbf{true} \vdash \neg\ wait' \wedge tr' = tr \wedge state' = state \oplus \{x \mapsto val(e, state)\})
\end{aligned}$$

□

4.4 Wait

The wait action simply allows a specific number of time units to pass before termination, and its reactive design is as follows, and the proof of the calculation from the original definition is straightforward and not given here.

Theorem 8.

$$Wait\ d \mathrel{\widehat{=}} \mathbf{R_{ct}}(\mathbf{true} \vdash {}^\frown\!/tr' = {}^\frown\!/tr \wedge state' = state \wedge (\#tr' - \#tr < d \lhd wait' \rhd \#tr' - \#tr = d))$$

A wait action never diverges because its precondition is **true**, and is in a waiting state if $wait'$ is **true** and the passed time since the start is less than d ($\#tr' - \#tr < d$), or terminates at time d. In addition, we have a new non-deterministic wait action that may wait for any time unit whose value is between d_1 and d_2.

$$Wait\ d_1..d_2 \mathrel{\widehat{=}} \mathbf{R_{ct}}\left(\mathbf{true} \vdash {}^\frown\!/tr' = {}^\frown\!/tr \between state' = state \wedge \left(\begin{array}{c}\#tr' - \#tr < d_2 \\ \lhd\ wait'\ \rhd \\ \left(\begin{array}{c}\#tr' - \#tr \geq d_1 \\ \wedge\ \#tr' - \#tr \leq d_2\end{array}\right)\end{array}\right)\right) \tag{4.51}$$

This definition states that an non-deterministic wait can terminate arbitrarily between d_1 and d_2. For example, the postcondition denotes that it is waiting only if the time is less than d_2, but can also terminate at any time between d_1 and d_2.

4.5 Conditional Choice and Guarded Action

The definitions of conditional choice and guarded action in the original *Circus Time* are given as follows

$$P \lhd b \rhd Q \mathrel{\widehat{=}} (b \wedge P) \vee (\neg\ b \wedge Q)$$
$$b \& P \mathrel{\widehat{=}} P \lhd b \rhd Stop$$

where b is a condition (but $tr, tr', ref, ref' \notin \alpha b$) and is included in the alphabet of P and Q. And the reactive designs of the two operators can be deduced from the above definitions.

Theorem 9. *Suppose P and Q are two Circus Time actions,*

$$P \lhd b \rhd Q = \mathbf{R_{ct}}((\neg\ P^f_f \lhd b \rhd \neg\ Q^f_f) \vdash (P^t_f \lhd b \rhd Q^t_f))$$

Proof.

$$\begin{aligned}
& P \lhd b \rhd Q && \text{[Theorem 1]}\\
=\ & \mathbf{R_{ct}}(\neg\ P^f_f \vdash P^t_f) \lhd b \rhd \mathbf{R_{ct}}(\neg\ Q^f_f \vdash Q^t_f) && \text{[Property B-L8,L9,L13]}\\
=\ & \mathbf{R_{ct}}((\neg\ P^f_f \vdash P^t_f) \lhd b \rhd (\neg\ Q^f_f \vdash Q^t_f)) && \text{[Property 1-L2]}\\
=\ & \mathbf{R_{ct}}((\neg\ P^f_f \lhd b \rhd \neg\ Q^f_f) \vdash (P^t_f \lhd b \rhd Q^t_f))
\end{aligned}$$

□

Theorem 10. *Suppose P is a Circus Time action,*

$$b\&P = \mathbf{R_{ct}}((b \Rightarrow \neg\, P_f^f) \vdash (P_f^t \lhd b \rhd (wait' \wedge \frown/tr' = \frown/tr)))$$

Proof.

$$\begin{array}{ll}
& b\&P & [\text{def-}\&] \\
= & P \lhd b \rhd Stop & [\text{Theorem 1 and } Stop] \\
= & \mathbf{R_{ct}}(\neg\, P_f^f \vdash P_f^t) \lhd b \rhd \mathbf{R_{ct}}(true \vdash wait' \wedge \frown/tr' = \frown/tr) & [\text{Property 1-L2}] \\
= & \mathbf{R_{ct}}((\neg\, P_f^f \lhd b \rhd true) \vdash (P_f^t \lhd b \rhd (wait' \wedge \frown/tr' = \frown/tr))) & [\text{prop. cal.}] \\
= & \mathbf{R_{ct}}((b \Rightarrow \neg\, P_f^f) \vdash (P_f^t \lhd b \rhd (wait' \wedge \frown/tr' = \frown/tr)))
\end{array}$$

□

4.6 Prefix

The prefix action $c.e \rightarrow P$ is usually constructed by a composition of a simple prefix and P itself, written as $(c.e \rightarrow Skip)\,;\, P$. Since the reactive design of sequential composition has been given in Section 4.2, we here need to deduce the reactive design of the simple prefix only. The definition (Definition 5) of the simple prefix in the original *Circus Time* is given as

$$c.e \rightarrow Skip \mathrel{\widehat{=}} \mathbf{CSP1_{ct}}\left(ok' \wedge \mathbf{R_{ct}}\left(\ wait\_com(c) \vee terminating\_com(c.e)\ \right)\right) \tag{4.52}$$

$$wait\_com(c) \mathrel{\widehat{=}} (wait' \wedge possible(ref, ref', c) \wedge \frown/tr' = \frown/tr) \tag{4.53}$$

$$possible(ref, ref', c) \mathrel{\widehat{=}} \forall\, i : \#ref..\#ref' \bullet c \notin ref'(i)$$

$$term\_com(c.e) \mathrel{\widehat{=}} (\neg\, wait' \wedge diff(tr', tr) = \langle\langle c.e \rangle\rangle) \tag{4.54}$$

$$terminating\_com(c.e) \mathrel{\widehat{=}} (term\_com(c.e) \vee wait\_com(c)\,;\, term\_com(c.e))$$

We construct the reactive design of the simple prefix as follows.

Lemma 18. $(c.e \rightarrow Skip)_f^f = \mathbf{R1_{ct}}(\neg\, ok)$

Proof.

$$\begin{array}{ll}
& (c.e \rightarrow Skip)_f^f & [\text{Definition 5}] \\
= & \left(\mathbf{CSP1_{ct}}\left(ok' \wedge \mathbf{R_{ct}}\left(\ wait_com(c) \vee terminating_com(c.e)\ \right)\right)\right)_f^f & [\mathbf{CSP1_{ct}}] \\
= & \left((\neg\, ok \wedge RT) \vee \left(ok' \wedge \mathbf{R_{ct}}\left(\ wait_com(c) \vee terminating_com(c.e)\ \right)\right)\right)_f^f & [\text{substitution}] \\
= & \left((\neg\, ok \wedge RT) \vee \left(\mathbf{false} \wedge \left(\mathbf{R_{ct}}\left(\ wait_com(c) \vee terminating_com(c.e)\ \right)\right)^f\right)\right)_f & [\text{prop. calculus}] \\
= & (\neg\, ok \wedge RT) \vee \mathbf{false} & [\text{propositional calculus}] \\
= & \neg\, ok \wedge RT & [\mathbf{R1_{ct}}] \\
= & \mathbf{R1_{ct}}(\neg\, ok)
\end{array}$$

□

Lemma 19.

$$(c.e \rightarrow Skip)_f^t = \mathbf{R1_{ct}}(\neg\, ok) \vee \mathbf{R1_{ct}}(wait\_com(c) \vee terminating\_com(c.e))$$

Proof.

$$\begin{array}{ll}
& (c.e \rightarrow Skip)_f^t & [\text{Definition 5}] \\
= & \left(\mathbf{CSP1_{ct}}\left(ok' \wedge \mathbf{R_{ct}}\left(\ wait_com(c) \vee terminating_com(c.e)\ \right)\right)\right)_f^t & [\mathbf{CSP1_{ct}}] \\
= & \left((\neg\, ok \wedge RT) \vee \left(ok' \wedge \mathbf{R_{ct}}\left(\ wait_com(c) \vee terminating_com(c.e)\ \right)\right)\right)_f^t & [\text{substitution}] \\
= & (\neg\, ok \wedge RT) \vee \left(\mathbf{true} \wedge \left(\mathbf{R_{ct}}\left(\ wait_com(c) \vee terminating_com(c.e)\ \right)\right)_f^t\right) & [\text{propositional calculus}]
\end{array}$$

$$
\begin{aligned}
&= (\neg\ ok \wedge RT) \vee \left(\mathbf{R_{ct}}\left(\ \ wait\_com(c) terminating\_com(c.e)\ \ \right)\right)^t_f && \text{[Law 24]}\\
&= (\neg\ ok \wedge RT) \vee \left(\mathbf{R1_{ct}} \circ \mathbf{R2_{ct}}\left(\left(\ \ wait\_com(c) \vee terminating\_com(c.e)\ \ \right)_f\right)\right)^t && \text{[substitution]}\\
&= (\neg\ ok \wedge RT) \vee \left(\mathbf{R1_{ct}} \circ \mathbf{R2_{ct}}\left(\left(\ \ wait\_com(c) \vee terminating\_com(c.e)\ \ \right)\right)\right) && \text{[Lemma 6,8 and } \mathbf{R1_{ct}}\text{]}\\
&= \mathbf{R1_{ct}}(\neg\ ok) \vee \mathbf{R1_{ct}}(wait\_com(c) \vee terminating\_com(c.e))
\end{aligned}
$$

□

Theorem 11.

$$c.e \rightarrow Skip \mathrel{\widehat{=}} \mathbf{R_{ct}}\left(\mathbf{true} \vdash \left(\ \ wait\_com(c) \vee terminating\_com(c.e)\ \ \right)\right)$$

Proof.

$$
\begin{aligned}
&c.e \rightarrow Skip && \text{[Theorem 1]}\\
&= \mathbf{R_{ct}}(\neg\ (c.e \rightarrow Skip)^f_f \vdash (c.e \rightarrow Skip)^t_f) && \text{[Lemma 18, 19]}\\
&= \mathbf{R_{ct}}\left(\neg\ \mathbf{R1_{ct}}(\neg\ ok) \vdash \mathbf{R1_{ct}}(\neg\ ok) \vee \mathbf{R1_{ct}}\left(\ wait\_com(c) \vee terminating\_com(c.e)\ \right)\right) && \text{[design]}\\
&= \mathbf{R_{ct}}\left(\begin{array}{c}\neg\ ok \vee \mathbf{R1_{ct}}(\neg\ ok) \vee (ok' \wedge \mathbf{R1_{ct}}(\neg\ ok)) \vee \\ \left(ok' \wedge \mathbf{R1_{ct}}\left(\ wait\_com(c) \vee terminating\_com(c.e)\ \right)\right)\end{array}\right) && \text{[Law 2 and propositional calculus]}\\
&= \mathbf{R_{ct}}(\neg\ ok \vee (ok' \wedge (wait\_com(c) \vee terminating\_com(c.e)))) && \text{[design]}\\
&= \mathbf{R_{ct}}(\mathbf{true} \vdash wait\_com(c) \vee terminating\_com(c.e))
\end{aligned}
$$

□

Note that the precondition in the design of the above definition is **true**, which means it never diverges. In a timed setting, a simple prefix can behave in three different ways. The clause $wait\_com(c)$ expresses that it can wait for interaction from its environment and meanwhile the channel c is not refusable. The clause $term\_com(c.e)$ simply denotes that the event is executed immediately. The composition of $wait\_com(c)$ and $term\_com(c.e)$ means that it may wait for a while and then terminates with a fired event $c.e$.

We have made a few changes to this definition for capturing the behaviour of a simple prefix more precisely. First, we constrain *state* and $state'$ in both $wait\_com(c)$ and $term\_com(c.e)$; otherwise the value of the local state can become arbitrary after that. Second, we constrain *ref* and ref' in $term\_com(c.e)$ in order to prevent it from losing the history of refusals. Finally, we change the view of considering the state at each time point. The interpretation of $wait\_com(c)$; $term\_com(c.e)$ is that, there is always a waiting state before the simple prefix terminates or executes the event $c.e$. In other words, this actually states that traces are prefix closed within the simple prefix, which is one of some important assumptions of the standard CSP models. In Section 5.2, we will use a counterexample to show that the *Circus Time* theory violates this assumption. The solution to this issue is that we allow the behaviour in $term\_com(c.e)$ to occur at next time unit after $wait\_com(c)$. To sum up, we give a new reactive design with regard to these changes.

Definition 10. *(***Simple prefix***)*

$$c.e \rightarrow Skip \mathrel{\widehat{=}} \mathbf{R_{ct}}(\mathbf{true} \vdash wait\_com(c) \vee term\_now\_com(c.e) \vee terminating\_com(c.e))$$

$$wait\_com(c) \mathrel{\widehat{=}} (wait' \wedge possible(ref, ref', c) \wedge \frown/tr' = \frown/tr \wedge state' = state) \tag{4.55}$$

$$possible(ref, ref', c) \mathrel{\widehat{=}} \forall\, i : \#ref \,..\, \#ref' \bullet c \notin ref'(i) \tag{4.56}$$

$$term\_now\_com(c.e) \mathrel{\widehat{=}} (\neg\ wait' \wedge front(ref') = front(ref) \wedge diff(tr', tr) = \langle\langle c.e\rangle\rangle \wedge state' = state) \tag{4.57}$$

$$term\_next\_com(c.e) \mathrel{\widehat{=}} (\neg\ wait' \wedge front(ref') = ref \wedge tr' - tr = \langle\langle c.e\rangle\rangle \wedge state' = state) \tag{4.58}$$

$$terminating\_com(c.e) \mathrel{\widehat{=}} (wait\_com(c)\ ;\ term\_next\_com(c.e)) \tag{4.59}$$

Note that the difference between $term\_now\_com(c.e)$ and $term\_next\_com(c.e)$ is that the former specifies that the event occurs in the current time unit (or the time unit in which the previous action terminates), while the latter specifies that it occurs in the next time unit.

If this new reactive design refines the one (Theorem 11) calculated from the original definition, and consequently it can be validated. In the theory of designs, a design, $(P_1 \vdash Q_1)$, refines or is stronger than another, if and only if, it has a weaker assumption, and has a stronger commitment, described as

Law 62. *(Refinement of designs)*

$$(P_2 \vdash Q_2 \sqsubseteq P_1 \vdash Q_1) = (P_1 \sqsubseteq P_2 \wedge Q_2 \sqsubseteq (P_2 \wedge Q_1))$$

Since the preconditions of both reactive designs are **true**, the refinement actually requires

$$(wait\_com(c)_O \vee terminating\_com(c.e)_O) \sqsubseteq (wait\_com(c) \vee term\_now\_com(c.e) \vee terminating\_com(c.e))$$

where we use the subscript O to denote that the predicate comes from the original definition.

Proof.

$$\begin{array}{ll}
& (wait_com(c) \vee term_now_com(c.e) \vee terminating_com(c.e)) \qquad [terminating_com(c.e)\ (4.59)] \\
= & (wait_com(c) \vee term_now_com(c.e) \vee (wait_com(c);\ term_next_com(c.e))) \quad [wait_com(c)\ (4.55)] \\
= & (wait' \wedge possible(ref, ref', c) \wedge \frown/tr' = \frown/tr \wedge state' = state) \vee \qquad [\text{propositional calculus}] \\
& term_now_com(c.e) \vee (wait_com(c);\ term_next_com(c.e))) \\
\Rightarrow & (wait' \wedge possible(ref, ref', c) \wedge \frown/tr' = \frown/tr) \vee \qquad [wait_com(c)_O (4.53)] \\
& term_now_com(c.e) \vee (wait_com(c);\ term_next_com(c.e))) \\
= & wait_com(c)_O \vee term_now_com(c.e) \vee (wait_com(c);\ term_next_com(c.e))) \\
& \qquad [term_now_com(c.e)(4.57)] \\
= & (\neg\ wait' \wedge front(ref') = front(ref) \wedge diff(tr', tr) = \langle\langle c.e \rangle\rangle \wedge state' = state) \vee \qquad [\text{propositional calculus}] \\
& wait_com(c)_O \vee (wait_com(c);\ term_next_com(c.e))) \\
\Rightarrow & wait_com(c)_O \vee (\neg\ wait' \wedge diff(tr', tr) = \langle\langle c.e \rangle\rangle) \vee (wait_com(c);\ term_next_com(c.e))) \\
& \qquad [term_com(c.e)_O\ (4.54)] \\
= & wait_com(c)_O \vee term_com(c.e)_O \vee (wait_com(c);\ term_next_com(c.e))) \\
& \qquad [wait_com(c)(4.55) \text{ and } term_next_com(c.e)(4.58)\ ] \\
= & wait_com(c)_O \vee term_com(c.e)_O \vee \qquad [\text{relational calculus}] \\
& \left(\begin{array}{c} (wait' \wedge possible(ref, ref', c) \wedge \frown/tr' = \frown/tr \wedge state' = state); \\ (\neg\ wait' \wedge front(ref') = ref \wedge tr' - tr = \langle\langle c.e \rangle\rangle \wedge state' = state) \end{array} \right) \\
= & wait_com(c)_O \vee term_com(c.e)_O \vee \qquad [\text{propositional calculus}] \\
& \left(\begin{array}{c} (wait' \wedge possible(ref, ref', c) \wedge \frown/tr' = \frown/tr);\ (\neg\ wait' \wedge tr' - tr = \langle\langle c.e \rangle\rangle) \wedge \\ ((possible(ref, ref', c);\ front(ref') = ref) \wedge state' = state) \end{array} \right) \\
\Rightarrow & wait_com(c)_O \vee term_com(c.e)_O \vee \qquad [\text{relational calculus}] \\
& (wait' \wedge possible(ref, ref', c) \wedge \frown/tr' = \frown/tr);\ (\neg\ wait' \wedge tr' - tr = \langle\langle c.e \rangle\rangle) \\
\Rightarrow & wait_com(c)_O \vee term_com(c.e)_O \vee \qquad [wait_com(c)_O \text{ and } term_com(c.e)_O\ ] \\
& (wait' \wedge possible(ref, ref', c) \wedge \frown/tr' = \frown/tr);\ (\neg\ wait' \wedge diff(tr', tr) = \langle\langle c.e \rangle\rangle) \\
= & wait_com(c)_O \vee term_com(c.e)_O \vee (wait_com(c)_O;\ term_com(c.e)_O) \qquad [terminating_com(c.e)] \\
= & wait_com(c)_O \vee terminating_com(c.e)_O
\end{array}$$

□

All in all then, the reactive design of a prefix like $c.e \rightarrow P$ can be calculated by the composition of a simple prefix and P itself.

Theorem 12. *(***Prefix***)*

$$c.e \rightarrow P = \mathbf{R_{ct}} \left(\begin{array}{c} \neg((term\_now\_com(c.e) \vee terminating\_com(c.e))\ ;\ \mathbf{R1_{ct}}(\neg\ wait \wedge \mathbf{R2_{ct}}(P_f^f))) \\ \vdash \\ (wait\_com(c) \vee term\_now\_com(c.e) \vee terminating\_com(c.e))\ ;\ \mathbf{R1_{ct}}(\mathbb{I} \lhd wait \rhd \mathbf{R2_{ct}}(P_f^t)) \end{array} \right)$$

Proof.

$$
\begin{array}{ll}
& c.e \rightarrow P & [\text{def of prefix}] \\
= & (c.e \rightarrow Skip);\ P & [\text{Theorem 6}] \\
= & \mathbf{R_{ct}} \left(\begin{array}{c} \neg\ (\mathbf{R1_{ct}}((c.e \rightarrow Skip)^f_f)\,;\, \mathbf{R1_{ct}}(\mathbf{true})) \wedge \neg\ (\mathbf{R1_{ct}}((c.e \rightarrow Skip)^t_f)\,;\, \mathbf{R1_{ct}}(\neg\ wait \wedge \mathbf{R2_{ct}}(P^f_f))) \\ \vdash \\ \mathbf{R1_{ct}}((c.e \rightarrow Skip)^t_f)\,;\, \mathbf{R1_{ct}}(\mathbb{I} \lhd wait \rhd \mathbf{R2_{ct}}(P^t_f)) \end{array} \right) & [\text{Theorem 11}] \\
= & \mathbf{R_{ct}} \left(\begin{array}{c} \neg\ (\mathbf{R1_{ct}}(\mathbf{false})\,;\, \mathbf{R1_{ct}}(\mathbf{true})) \wedge \neg\ (\mathbf{R1_{ct}}((c.e \rightarrow Skip)^t_f)\,;\, \mathbf{R1_{ct}}(\neg\ wait \wedge \mathbf{R2_{ct}}(P^f_f))) \\ \vdash \\ \mathbf{R1_{ct}}((c.e \rightarrow Skip)^t_f)\,;\, \mathbf{R1_{ct}}(\mathbb{I} \lhd wait \rhd \mathbf{R2_{ct}}(P^t_f)) \end{array} \right) & [\text{Theorem 11 and relational calculus}] \\
= & \mathbf{R_{ct}} \left(\begin{array}{c} \neg(\mathbf{R1_{ct}}(wait\_com(c) \vee term\_now\_com(c.e) \vee terminating\_com(c.e))\,;\, \mathbf{R1_{ct}}(\neg\, wait \wedge \mathbf{R2_{ct}}(P^f_f))) \\ \vdash \\ \mathbf{R1_{ct}}(wait\_com(c) \vee term\_now\_com(c.e) \vee terminating\_com(c.e))\,;\, \mathbf{R1_{ct}}(\mathbb{I} \lhd wait \rhd \mathbf{R2_{ct}}(P^t_f)) \end{array} \right) & [\text{relational calculus}] \\
= & \mathbf{R_{ct}} \left(\begin{array}{c} \neg(term\_now\_com(c.e) \vee terminating\_com(c.e))\,;\, \mathbf{R1_{ct}}(\neg\, wait \wedge \mathbf{R2_{ct}}(P^f_f))) \\ \vdash \\ (wait\_com(c) \vee term\_now\_com(c.e) \vee terminating\_com(c.e))\,;\, \mathbf{R1_{ct}}(\mathbb{I} \lhd wait \rhd \mathbf{R2_{ct}}(P^t_f)) \end{array} \right) &
\end{array}
$$

□

This theorem states that if it does not hold that P diverges after the termination of $c.e \rightarrow Skip$, this prefix just behaves either like $wait\_com(c)$ or like the composition of the execution of $c.e$ and P^t_f.

4.7 Internal Choice

The internal choice is totally out of control of its environment, and its definition in the original *Circus Time* is identical to that in the UTP.

$$P \sqcap Q \mathrel{\widehat{=}} P \vee Q$$

The reactive design of internal choice can be easily deduced as follows.

Theorem 13. *Suppose P and Q are two Circus Time actions,*

$$P \sqcap Q = \mathbf{R_{ct}}((\neg\ P^f_f \wedge \neg\ Q^f_f) \vdash (P^t_f \vee Q^t_f))$$

This states that if both P and Q do not diverge, it then behaves like either P^t_f or Q^t_f.

Proof.

$$
\begin{array}{ll}
& P \sqcap Q & [\text{def of } \sqcap] \\
= & P \vee Q & [\text{Theorem 1}] \\
= & \mathbf{R_{ct}}(\neg\ P^f_f \vdash P^t_f) \vee \mathbf{R_{ct}}(\neg\ Q^f_f \vdash Q^t_f) & [\text{Law 2,8,15}] \\
= & \mathbf{R_{ct}}((\neg\ P^f_f \vdash P^t_f) \vee (\neg\ Q^f_f \vdash Q^t_f)) & [\text{Property 1-L1}] \\
= & \mathbf{R_{ct}}((\neg\ P^f_f \wedge \neg\ Q^f_f) \vdash (P^t_f \vee Q^t_f)) &
\end{array}
$$

□

4.8 External Choice

The action $P \,\Box\, Q$ may behave either like the conjunction of P and Q if no external event has been observed yet, or like their disjunction if the decision has been made. In a timed environment, this action in fact requires that any waiting must be agreed by both actions. For example, the action, *Wait* 2 $\Box$ *Wait* 3, can behave only like *Wait* 2 because the waiting observation after two time units cannot be satisfied by *Wait* 2. Moreover, *Circus Time* records the whole history of refusals throughout an execution rather than at the end. In other words, other than the agreement on the traces, the refusals must also be agreed by both actions in any waiting period. For example, if the event a resolves the external choice in $a \to P \,\Box\, b \to Q$, unlike the approach in CSP, we cannot just use $a \to P$ to express the behaviour. This action can also not refuse b before the occurrence of a.

On the face of it, the external choice in *Circus Time* is far more complex than that in CSP. As usual, we apply Theorem 1 to the definition in the original *Circus Time* and then obtain the following reactive design.

Lemma 20. $DifDetected(P, Q)^f_f = \mathbf{true}$

Proof.

$$
\begin{array}{ll}
& DifDetected(P, Q)^f_f \qquad \text{[Definition 7]} \\
= & \left(\begin{array}{l} \neg\, ok' \vee \left(\begin{array}{l} (ok \wedge \neg\, wait \wedge ((P \wedge Q \wedge ok' \wedge wait' \wedge {}^\frown\!/tr' = {}^\frown\!/tr) \vee Skip))\,; \\ \left(\begin{array}{c} (ok' \wedge \neg\, wait' \wedge tr' = tr \wedge ref' = ref \wedge \#tr = \#ref \wedge \#tr' = \#ref') \\ \vee\, (ok' \wedge head(diff(tr', tr)) \neq \langle\rangle) \end{array} \right) \end{array} \right) \end{array} \right)^f_f \\
& \qquad \text{[substitution]} \\
= & \left(\begin{array}{l} \neg\, \mathbf{false} \vee \left(\begin{array}{l} (ok \wedge \neg\, \mathbf{false} \wedge ((P^f_f \wedge Q^f_f \wedge ok' \wedge wait' \wedge {}^\frown\!/tr' = {}^\frown\!/tr) \vee Skip^f_f))\,; \\ \left(\begin{array}{c} (\mathbf{false} \wedge \neg\, wait' \wedge tr' = tr \wedge ref' = ref \wedge \#tr = \#ref \wedge \#tr' = \#ref') \\ \vee\, (\mathbf{false} \wedge head(diff(tr', tr)) \neq \langle\rangle) \end{array} \right) \end{array} \right) \end{array} \right) \\
& \qquad \text{[relational calculus]} \\
= & \mathbf{true}
\end{array}
$$

□

Lemma 21.

$$DifDetected(P, Q)^t_f = ((ok \wedge P^t_f \wedge Q^t_f \wedge wait' \wedge {}^\frown\!/tr' = {}^\frown\!/tr);\ termination) \vee termination$$

$$termination = (\neg\, wait' \wedge tr' = tr) \vee (head(diff(tr', tr)) \neq \langle\rangle)$$

Proof.

$$
\begin{array}{ll}
& DifDetected(P, Q)^t_f \qquad \text{[Definition 7]} \\
= & \left(\begin{array}{l} \neg\, ok' \vee \left(\begin{array}{l} (ok \wedge \neg\, wait \wedge ((P \wedge Q \wedge ok' \wedge wait' \wedge {}^\frown\!/tr' = {}^\frown\!/tr) \vee Skip))\,; \\ \left(\begin{array}{c} (ok' \wedge \neg\, wait' \wedge tr' = tr \wedge ref' = ref \wedge \#tr = \#ref \wedge \#tr' = \#ref') \\ \vee\, (ok' \wedge head(diff(tr', tr)) \neq \langle\rangle) \end{array} \right) \end{array} \right) \end{array} \right)^t_f \qquad \text{[subs.]} \\
= & \left(\begin{array}{l} \neg\, \mathbf{true} \vee \left(\begin{array}{l} (ok \wedge \neg\, \mathbf{false} \wedge ((P^t_f \wedge Q^t_f \wedge \mathbf{true} \wedge wait' \wedge {}^\frown\!/tr' = {}^\frown\!/tr) \vee Skip^t_f))\,; \\ \left(\begin{array}{c} (\mathbf{true} \wedge \neg\, wait' \wedge tr' = tr \wedge ref' = ref \wedge \#tr = \#ref \wedge \#tr' = \#ref') \\ \vee\, (\mathbf{true} \wedge head(diff(tr', tr)) \neq \langle\rangle) \end{array} \right) \end{array} \right) \end{array} \right) \\
& \qquad \text{[propositional calculus]} \\
= & \left(\begin{array}{l} (ok \wedge ((P^t_f \wedge Q^t_f \wedge wait' \wedge {}^\frown\!/tr' = {}^\frown\!/tr) \vee Skip^t_f))\,; \\ ((\neg\, wait' \wedge tr' = tr \wedge ref' = ref \wedge \#tr = \#ref \wedge \#tr' = \#ref') \vee (head(diff(tr', tr)) \neq \langle\rangle)) \end{array} \right) \\
& \qquad \text{[relational calculus]} \\
= & \left(\begin{array}{l} (ok \wedge P^t_f \wedge Q^t_f \wedge wait' \wedge {}^\frown\!/tr' = {}^\frown\!/tr);\ (\neg\, wait' \wedge tr' = tr \wedge ref' = ref \wedge \#tr = \#ref \wedge \#tr' = \#ref') \vee \\ ((ok \wedge P^t_f \wedge Q^t_f \wedge wait' \wedge {}^\frown\!/tr' = {}^\frown\!/tr);\ head(diff(tr', tr)) \neq \langle\rangle) \vee \\ (ok \wedge Skip^t_f);\ (\neg\, wait' \wedge tr' = tr \wedge ref' = ref \wedge \#tr = \#ref \wedge \#tr' = \#ref') \vee \\ (ok \wedge Skip^t_f);\ head(diff(tr', tr)) \neq \langle\rangle \end{array} \right) \\
& \qquad \text{[relational calculus]}
\end{array}
$$

$$= \begin{pmatrix} (ok \wedge P_f^t \wedge Q_f^t \wedge wait' \wedge \frown/tr' = \frown/tr);\ (\neg\ wait' \wedge tr' = tr) \vee \\ ((ok \wedge P_f^t \wedge Q_f^t \wedge wait' \wedge \frown/tr' = \frown/tr);\ head(diff(tr', tr)) \neq \langle\rangle) \vee \\ (ok \wedge Skip_f^t);\ (\neg\ wait' \wedge tr' = tr \wedge ref' = ref \wedge \#tr = \#ref \wedge \#tr' = \#ref') \vee \\ (ok \wedge Skip_f^t);\ head(diff(tr', tr)) \neq \langle\rangle \end{pmatrix} \quad \text{[Lemma 12]}$$

$$= \begin{pmatrix} (ok \wedge P_f^t \wedge Q_f^t \wedge wait' \wedge \frown/tr' = \frown/tr);\ (\neg\ wait' \wedge tr' = tr) \vee \\ ((ok \wedge P_f^t \wedge Q_f^t \wedge wait' \wedge \frown/tr' = \frown/tr);\ head(diff(tr', tr)) \neq \langle\rangle) \vee \\ \left(\begin{array}{l} (ok \wedge (\mathbf{R1_{ct}}(\neg\ ok) \vee \mathbf{R1_{ct}}(\neg\ wait' \wedge tr' = tr \wedge state' = state)));\ \\ (\neg\ wait' \wedge tr' = tr \wedge ref' = ref \wedge \#tr = \#ref \wedge \#tr' = \#ref') \end{array}\right) \vee \\ (ok \wedge (\mathbf{R1_{ct}}(\neg\ ok) \vee \mathbf{R1_{ct}}(\neg\ wait' \wedge tr' = tr \wedge state' = state)));\ head(diff(tr', tr)) \neq \langle\rangle \end{pmatrix}$$

[relational calculus]

$$= \begin{pmatrix} (ok \wedge P_f^t \wedge Q_f^t \wedge wait' \wedge \frown/tr' = \frown/tr);\ (\neg\ wait' \wedge tr' = tr) \vee \\ ((ok \wedge P_f^t \wedge Q_f^t \wedge wait' \wedge \frown/tr' = \frown/tr);\ head(diff(tr', tr)) \neq \langle\rangle) \vee \\ (\neg\ wait' \wedge tr' = tr) \vee (head(diff(tr', tr)) \neq \langle\rangle) \end{pmatrix}$$

[Let $termination = (\neg\ wait' \wedge tr' = tr) \vee (head(diff(tr', tr)) \neq \langle\rangle)$]

$$= ((ok \wedge P_f^t \wedge Q_f^t \wedge wait' \wedge \frown/tr' = \frown/tr);\ termination) \vee termination$$

□

Lemma 22. $(P \,\square\, Q)_f^f = P_f^f \vee Q_f^f$

Proof.

$$\begin{array}{ll} & (P \,\square\, Q)_f^f & \text{[Definition 7]} \\ = & (\mathbf{CSP2_{ct}}((P \wedge Q \wedge Stop) \vee (DifDetected(P, Q) \wedge (P \vee Q))))_f^f & [\mathbf{CSP2_{ct}} \text{ and Law 40}] \\ = & \left(\begin{array}{l} ((P \wedge Q \wedge Stop) \vee (DifDetected(P, Q) \wedge (P \vee Q)))^f \vee \\ (ok' \wedge ((P \wedge Q \wedge Stop) \vee (DifDetected(P, Q) \wedge (P \vee Q)))^t) \end{array}\right)_f^f & \text{[substitution on } ok'] \\ = & ((P \wedge Q \wedge Stop) \vee (DifDetected(P, Q) \wedge (P \vee Q)))_f^f & \text{[substitution]} \\ = & (P_f^f \wedge Q_f^f \wedge Stop_f^f) \vee (DifDetected(P, Q)_f^f \wedge (P_f^f \vee Q_f^f)) & \text{[Lemma 13,20]} \\ = & (P_f^f \wedge Q_f^f \wedge \mathbf{R1_{ct}}(\neg\ ok)) \vee (\mathbf{true} \wedge (P_f^f \vee Q_f^f)) & \text{[propositional calculus]} \\ = & P_f^f \vee Q_f^f & \end{array}$$

□

Lemma 23.

$$(P \,\square\, Q)_f^t = \mathbf{R1_{ct}}(\neg\ ok) \vee (P_f^t \wedge Q_f^t \wedge wait' \wedge \frown/tr' = \frown/tr \wedge state' = state) \vee (Diff(P_f^t, Q_f^t) \wedge (P_f^t \vee Q_f^t))$$

$$Diff(P, Q) = ((P \wedge Q \wedge wait' \wedge \frown/tr' = \frown/tr);\ termination) \vee termination$$

Proof.

$$
\begin{aligned}
& (P \Box Q)^t_f && \text{[Definition 7]}\\
=& (\mathbf{CSP2_{ct}}((P \wedge Q \wedge Stop) \vee (DifDetected(P,Q) \wedge (P \vee Q))))^t_f && [\mathbf{CSP2_{ct}} \text{ and Law 40}]\\
=& \left(\begin{array}{l} ((P \wedge Q \wedge Stop) \vee (DifDetected(P,Q) \wedge (P \vee Q)))^f \vee \\ (ok' \wedge ((P \wedge Q \wedge Stop) \vee (DifDetected(P,Q) \wedge (P \vee Q)))^t) \end{array} \right)^t_f && \text{[substitution on } ok']\\
=& ((P \wedge Q \wedge Stop) \vee (DifDetected(P,Q) \wedge (P \vee Q)))^t_f && \text{[substitution]}\\
=& (P^t_f \wedge Q^t_f \wedge Stop^t_f) \vee (DifDetected(P,Q)^t_f \wedge (P^t_f \vee Q^t_f)) && \text{[Lemma 13,20]}\\
=& (P^t_f \wedge Q^t_f \wedge (\mathbf{R1_{ct}}(\neg\, ok) \vee (wait' \wedge \frown/tr' = \frown/tr))) \vee\\
& ((((ok \wedge P^t_f \wedge Q^t_f \wedge wait' \wedge \frown/tr' = \frown/tr);\, termination) \vee termination) \wedge (P^t_f \vee Q^t_f)) && [P \text{ is } \mathbf{CSP1_{ct}}]\\
=& \mathbf{R1_{ct}}(\neg\, ok) \vee (P^t_f \wedge Q^t_f \wedge wait' \wedge \frown/tr' = \frown/tr) \vee\\
& ((((ok \wedge P^t_f \wedge Q^t_f \wedge wait' \wedge \frown/tr' = \frown/tr);\, termination) \vee termination) \wedge (P^t_f \vee Q^t_f))\\
& \qquad [\text{Let } Diff(P,Q) = ((P \wedge Q \wedge wait' \wedge \frown/tr' = \frown/tr);\, termination) \vee termination\ ]\\
=& \mathbf{R1_{ct}}(\neg\, ok) \vee (P^t_f \wedge Q^t_f \wedge wait' \wedge \frown/tr' = \frown/tr) \vee (Diff(P^t_f, Q^t_f) \wedge (P^t_f \vee Q^t_f))
\end{aligned}
$$

□

Theorem 14.

$$P \Box Q = \mathbf{R_{ct}}(\neg\, P^f_f \wedge \neg\, Q^f_f \vdash (P^t_f \wedge Q^t_f \wedge wait' \wedge \frown/tr' = \frown/tr) \vee (Diff(P^t_f, Q^t_f) \wedge (P^t_f \vee Q^t_f)))$$

$$Diff(P,Q) \mathrel{\widehat{=}} ((P \wedge Q \wedge wait' \wedge \frown/tr' = \frown/tr);\, termination) \vee termination$$

$$termination \mathrel{\widehat{=}} (\neg\, wait' \wedge tr' = tr) \vee head(diff(tr', tr)) \neq \langle\rangle$$

Proof.

$$
\begin{aligned}
& P \Box Q && \text{[Theorem 1]}\\
=& \mathbf{R_{ct}}(\neg\, (P \Box Q)^f_f \vdash (P \Box Q)^t_f) && \text{[design]}\\
=& \mathbf{R_{ct}}(\neg\, ok \vee (P \Box Q)^f_f \vee (ok' \wedge (P \Box Q)^t_f))) && \text{[Lemma 22,23]}\\
=& \mathbf{R_{ct}} \left(\begin{array}{l} \neg\, ok \wedge P^f_f \vee Q^f_f \vee \\ (ok' \wedge (\mathbf{R1_{ct}}(\neg\, ok) \vee (P^t_f \wedge Q^t_f \wedge wait' \wedge \frown/tr' = \frown/tr) \vee (Diff(P^t_f, Q^t_f) \wedge (P^f_f \vee Q^f_f)))) \end{array} \right)\\
& && \text{[propositional calculus]}\\
=& \mathbf{R_{ct}} \left(\begin{array}{l} \neg\, ok \wedge P^f_f \vee Q^f_f \vee \\ (ok' \wedge ((P^t_f \wedge Q^t_f \wedge wait' \wedge \frown/tr' = \frown/tr) \vee (Diff(P^t_f, Q^t_f) \wedge (P^t_f \vee Q^t_f)))) \end{array} \right) && \text{[design]}\\
=& \mathbf{R_{ct}}(\neg\, P^f_f \wedge \neg\, Q^f_f \vdash (P^t_f \wedge Q^t_f \wedge wait' \wedge \frown/tr' = \frown/tr) \vee (Diff(P^t_f, Q^t_f) \wedge (P^t_f \vee Q^t_f)))
\end{aligned}
$$

□

This reactive design states that if neither of P and Q diverges, its postcondition reveals all cases in the external choice: P and Q are agreed on waiting without executing any observable event, or in the beginning they are agreed on waiting and then terminate (the former clause in *termination*) or execute external events (the latter clause in *termination*), or they simply behave like *termination* immediately.

With regard to this reactive design, we mainly adopt two changes to its pre and postcondition respectively. For the change in its postcondition, we redefine $Diff(P,Q)$ as

$$Diff(P,Q) \mathrel{\widehat{=}} (((P \wedge Q \wedge wait' \wedge \frown/tr' = \frown/tr)\,;\, term\_next) \vee term\_now) \tag{4.60}$$

$$term\_now \mathrel{\widehat{=}} ((\neg\, wait' \wedge tr' = tr) \vee head(diff(tr', tr)) \neq \langle\rangle) \tag{4.61}$$

$$term\_next \mathrel{\widehat{=}} ((\neg\, wait' \wedge tr' - tr = \langle\langle\rangle\rangle) \vee head(tr' - tr) \neq \langle\rangle) \tag{4.62}$$

Clearly, the difference between *term_now* and *term_next* is the same as that between *term_now_com* and *term_next_com*, which is explained in Section 4.6. The predicate *term_now* denotes the immediate termination or execution of observable events, whereas *term_next* states that the same actions occur at next time

unit. Without this change, we cannot prove one of useful laws, $(Wait\ m \Box Wait\ m+n) = Wait\ m$. For the change in the precondition of Theorem 14, we, here, adopt a different approach to deal with divergences.

We discern that the behaviour before a divergence captured in Theorem 14 of the original *Circus Time* theory is simply but not very accurate. For example, the original definition of external choice uses $P_f^f \vee Q_f^f$ to capture the divergent behaviour individually. However, it introduces extra observation in an external choice, which may result in false verification of properties. For example, $(Wait\ 3 ; Chaos) \Box Wait\ 2$ should terminate after two time units and never diverge, while it has a divergent state in the original *Circus Time* theory. Therefore, in this new theory, we change the precondition of external choice by considering the divergent cases individually.

In a reactive design, the case for the divergence of the external choice's predecessor is captured by $\mathbf{R3_{ct}}$, and the divergent cases for itself are presented in the precondition

$$\neg\,(((P_f^f \vee Q_f^f) \wedge tr' = tr)\,;\, \mathbf{R1_{ct}}(\mathbf{true})) \wedge \tag{4.63}$$

$$\neg\,((P_f^f \wedge ((Q_f \wedge \frown/tr' = \frown/tr \wedge wait');\, tr' - tr = \langle\langle\rangle\rangle));\, \mathbf{R1_{ct}}(\mathbf{true})) \wedge \tag{4.64}$$

$$\neg\,((Q_f^f \wedge ((P_f \wedge \frown/tr' = \frown/tr \wedge wait');\, tr' - tr = \langle\langle\rangle\rangle));\, \mathbf{R1_{ct}}(\mathbf{true})) \wedge \tag{4.65}$$

$$\neg\,(P_f^f \wedge ((Q_f \wedge \frown/tr' = \frown/tr \wedge wait')\,;\, head(tr' - tr) \neq \langle\rangle)) \wedge \tag{4.66}$$

$$\neg\,(Q_f^f \wedge ((P_f \wedge \frown/tr' = \frown/tr \wedge wait')\,;\, head(tr' - tr) \neq \langle\rangle)) \wedge \tag{4.67}$$

$$\neg\,((P_f^f \vee Q_f^f) \wedge head(diff(tr', tr)) \neq \langle\rangle) \tag{4.68}$$

where, first, the predicate (4.63) states the external choice becomes divergent if any of them diverges immediately, in which the behaviour after the divergence is simply $\mathbf{R1_{ct}}(\mathbf{true})$; second, (4.64) and (4.65) state that either P or Q diverges, but both of them have not executed any event; third, (4.66) and (4.67) state that either of them experiences an agreement on waiting and a subsequent execution of external events before the divergence; finally, they immediately execute the external events and diverge later in (4.68). Note that we always judge a divergence one time unit early so as to avoid *Miracle*. That is the reason that we use $tr' - tr = \langle\langle\rangle\rangle$ and $head(tr' - tr) \neq \langle\rangle$ after each agreement on waiting. The entire reactive design of the external choice operator is given in the following theorem

Theorem 15. *(***External choice***)*

$$P \Box Q \mathrel{\widehat{=}} \mathbf{R_{ct}} \left(\begin{array}{c} \neg\,(((P_f^f \vee Q_f^f) \wedge tr' = tr)\,;\, \mathbf{R1_{ct}}(\mathbf{true})) \wedge \\ \neg\,((P_f^f \wedge ((Q_f \wedge \frown/tr' = \frown/tr \wedge wait');\, tr' - tr = \langle\langle\rangle\rangle));\, \mathbf{R1_{ct}}(\mathbf{true})) \wedge \\ \neg\,((Q_f^f \wedge ((P_f \wedge \frown/tr' = \frown/tr \wedge wait');\, tr' - tr = \langle\langle\rangle\rangle));\, \mathbf{R1_{ct}}(\mathbf{true})) \wedge \\ \neg\,(P_f^f \wedge ((Q_f \wedge \frown/tr' = \frown/tr \wedge wait')\,;\, head(tr' - tr) \neq \langle\rangle)) \wedge \\ \neg\,(Q_f^f \wedge ((P_f \wedge \frown/tr' = \frown/tr \wedge wait')\,;\, head(tr' - tr) \neq \langle\rangle)) \wedge \\ \neg\,((P_f^f \vee Q_f^f) \wedge head(diff(tr', tr)) \neq \langle\rangle) \\ \vdash \\ (P_f^t \wedge Q_f^t \wedge wait' \wedge \frown/tr' = \frown/tr) \vee (Diff(P_f^t, Q_f^t) \wedge (P_f^t \vee Q_f^t)) \end{array} \right)$$

Here, the definition of *Diff* is in (4.60). Although this reactive design of external choice seems a little bit complicated, it is able to accurately capture the divergent behaviour. In addition, this definition holds all algebraic laws about external choice like $P \Box Chaos = Chaos$ in the original model. In addition, this new reactive design can also refine the original one (Theorem 14). To validate this definition, we only need to prove the following two refinement relations

$$\left(\begin{array}{c} (P_f^t \wedge Q_f^t \wedge wait' \wedge \frown/tr = \frown/tr);\, termination) \\ \sqsubseteq \\ (\neg\, P_f^f \wedge \neg\, Q_f^f \wedge ((P_f^t \wedge Q_f^t \wedge wait' \wedge \frown/tr = \frown/tr);\, term\_next)) \end{array} \right) \tag{4.69}$$

Proof.

$$
\begin{aligned}
&\neg\, P_f^f \wedge \neg\, Q_f^f \wedge ((P_f^t \wedge Q_f^t \wedge wait' \wedge {}^\frown/tr = {}^\frown/tr);\, term\_next) && \text{[propositional calculus]}\\
\Rightarrow &(P_f^t \wedge Q_f^t \wedge wait' \wedge {}^\frown/tr = {}^\frown/tr);\, term\_next && [term\_next]\\
= &(P_f^t \wedge Q_f^t \wedge wait' \wedge {}^\frown/tr = {}^\frown/tr);\, ((\neg\, wait' \wedge tr' - tr = \langle\langle\rangle\rangle) \vee head(tr' - tr) \neq \langle\rangle) && \text{[rel. calculus]}\\
\Rightarrow &(P_f^t \wedge Q_f^t \wedge wait' \wedge {}^\frown/tr = {}^\frown/tr);\, ((\neg\, wait' \wedge tr' = tr) \vee head(diff(tr', tr)) \neq \langle\rangle) && [termination]\\
= &(P_f^t \wedge Q_f^t \wedge wait' \wedge {}^\frown/tr = {}^\frown/tr);\, termination
\end{aligned}
$$

□

$$
\left(\begin{array}{c}
\neg\, (((P_f^f \vee Q_f^f) \wedge tr' = tr)\,;\, \mathbf{R1_{ct}}(\mathbf{true})) \wedge\\
\neg\, ((P_f^f \wedge ((Q_f \wedge {}^\frown/tr' = {}^\frown/tr \wedge wait');\, tr' - tr = \langle\langle\rangle\rangle));\, \mathbf{R1_{ct}}(\mathbf{true})) \wedge\\
\neg\, ((Q_f^f \wedge ((P_f \wedge {}^\frown/tr' = {}^\frown/tr \wedge wait');\, tr' - tr = \langle\langle\rangle\rangle));\, \mathbf{R1_{ct}}(\mathbf{true})) \wedge\\
\neg\, (P_f^f \wedge ((Q_f \wedge {}^\frown/tr' = {}^\frown/tr \wedge wait')\,;\, head(tr' - tr) \neq \langle\rangle)) \wedge\\
\neg\, (Q_f^f \wedge ((P_f \wedge {}^\frown/tr' = {}^\frown/tr \wedge wait')\,;\, head(tr' - tr) \neq \langle\rangle)) \wedge\\
\neg\, ((P_f^f \vee Q_f^f) \wedge head(diff(tr', tr)) \neq \langle\rangle)
\end{array}\right) \sqsubseteq \neg\, P_f^f \wedge \neg\, Q_f^f \qquad (4.70)
$$

Proof. The above refinement relation depends on the following seven implication relations.

(1). $((P_f^f \wedge tr' = tr)\,;\, \mathbf{R1_{ct}}(\mathbf{true})) \Rightarrow P_f^f \vee Q_f^f$

$$
\begin{aligned}
&((P_f^f \wedge tr' = tr)\,;\, \mathbf{R1_{ct}}(\mathbf{true})) && \text{[relational calculus]}\\
\Rightarrow &P_f^f;\ \mathbf{R1_{ct}}(\mathbf{true}) && [P \text{ is } \mathbf{CSP4_{ct}}]\\
= &P_f^f && \text{[propositional calculus]}\\
\Rightarrow &P_f^f \vee Q_f^f
\end{aligned}
$$

(2). $((Q_f^f \wedge tr' = tr)\,;\, \mathbf{R1_{ct}}(\mathbf{true})) \Rightarrow P_f^f \vee Q_f^f$

The proof is similar to (1)

(3). $(P_f^f \wedge ((Q_f \wedge {}^\frown/tr' = {}^\frown/tr \wedge wait');\, tr' - tr = \langle\langle\rangle\rangle));\, \mathbf{R1_{ct}}(\mathbf{true}) \Rightarrow P_f^f \vee Q_f^f$

$$
\begin{aligned}
&(P_f^f \wedge ((Q_f \wedge {}^\frown/tr' = {}^\frown/tr \wedge wait');\, tr' - tr = \langle\langle\rangle\rangle));\, \mathbf{R1_{ct}}(\mathbf{true}) && \text{[relational caluclus]}\\
\Rightarrow &P_f^f;\, \mathbf{R1_{ct}}(\mathbf{true}) && [P \text{ is } \mathbf{CSP4_{ct}}]\\
= &P_f^f && \text{[propositional calculus]}\\
\Rightarrow &P_f^f \vee Q_f^f
\end{aligned}
$$

(4). $(Q_f^f \wedge ((P_f \wedge {}^\frown/tr' = {}^\frown/tr \wedge wait');\, tr' - tr = \langle\langle\rangle\rangle));\, \mathbf{R1_{ct}}(\mathbf{true}) \Rightarrow P_f^f \vee Q_f^f$

The proof is similar to (3)

(5). $P_f^f \wedge ((Q_f \wedge {}^\frown/tr' = {}^\frown/tr \wedge wait')\,;\, head(tr' - tr) \neq \langle\rangle) \Rightarrow P_f^f \vee Q_f^f$

(6). $Q_f^f \wedge ((P_f \wedge {}^\frown/tr' = {}^\frown/tr \wedge wait')\,;\, head(tr' - tr) \neq \langle\rangle) \Rightarrow P_f^f \vee Q_f^f$

(7). $(P_f^f \vee Q_f^f) \wedge head(diff(tr', tr)) \neq \langle\rangle \Rightarrow P_f^f \vee Q_f^f$

The proof of (5),(6) and (7) are straightforward.

□

Law 63. $P \,\Box\, Chaos = Chaos$

Proof.

$$
\begin{aligned}
& P \,\Box\, Chaos \\
=& \mathbf{R_{ct}} \left(\begin{array}{c}
\neg\,(((P_f^f \vee Chaos_f^f) \wedge tr' = tr)\,;\, \mathbf{R1_{ct}}(\mathbf{true})) \wedge \\
\neg\,((P_f^f \wedge ((Chaos_f \wedge \frown/tr' = \frown/tr \wedge wait')\,;\, tr' - tr = \langle\langle\rangle\rangle))\,;\, \mathbf{R1_{ct}}(\mathbf{true})) \wedge \\
\neg\,((Chaos_f^f \wedge ((P_f \wedge \frown/tr' = \frown/tr \wedge wait')\,;\, tr' - tr = \langle\langle\rangle\rangle))\,;\, \mathbf{R1_{ct}}(\mathbf{true})) \wedge \\
\neg\,(P_f^f \wedge ((Chaos_f \wedge \frown/tr' = \frown/tr \wedge wait')\,;\, head(tr' - tr) \neq \langle\rangle)) \wedge \\
\neg\,(Chaos_f^f \wedge ((P_f \wedge \frown/tr' = \frown/tr \wedge wait')\,;\, head(tr' - tr) \neq \langle\rangle)) \wedge \\
\neg\,((P_f^f \vee Chaos_f^f) \wedge head(diff(tr', tr)) \neq \langle\rangle) \\
\vdash \\
(P_f^t \wedge Chaos_f^t \wedge wait' \wedge \frown/tr' = \frown/tr) \vee (Diff(P_f^t, Chaos_f^t) \wedge (P_f^t \vee Chaos_f^t))
\end{array} \right) \\
& \qquad [Chaos(\text{Theorem 4})] \\
=& \mathbf{R_{ct}} \left(\begin{array}{c}
\neg\,(((P_f^f \vee \mathbf{true}) \wedge tr' = tr)\,;\, \mathbf{R1_{ct}}(\mathbf{true})) \wedge \\
\neg\,((P_f^f \wedge ((Chaos_f \wedge \frown/tr' = \frown/tr \wedge wait')\,;\, tr' - tr = \langle\langle\rangle\rangle))\,;\, \mathbf{R1_{ct}}(\mathbf{true})) \wedge \\
\neg\,((Chaos_f^f \wedge ((P_f \wedge \frown/tr' = \frown/tr \wedge wait')\,;\, tr' - tr = \langle\langle\rangle\rangle))\,;\, \mathbf{R1_{ct}}(\mathbf{true})) \wedge \\
\neg\,(P_f^f \wedge ((Chaos_f \wedge \frown/tr' = \frown/tr \wedge wait')\,;\, head(tr' - tr) \neq \langle\rangle)) \wedge \\
\neg\,(Chaos_f^f \wedge ((P_f \wedge \frown/tr' = \frown/tr \wedge wait')\,;\, head(tr' - tr) \neq \langle\rangle)) \wedge \\
\neg\,((P_f^f \vee Chaos_f^f) \wedge head(diff(tr', tr)) \neq \langle\rangle) \\
\vdash \\
(P_f^t \wedge Chaos_f^t \wedge wait' \wedge \frown/tr' = \frown/tr) \vee (Diff(P_f^t, Chaos_f^t) \wedge (P_f^t \vee Chaos_f^t))
\end{array} \right) \\
& \qquad [\text{relational calculus}] \\
=& \mathbf{R_{ct}} \left(\begin{array}{c}
\mathbf{false} \wedge \\
\neg\,((P_f^f \wedge ((Chaos_f \wedge \frown/tr' = \frown/tr \wedge wait')\,;\, tr' - tr = \langle\langle\rangle\rangle))\,;\, \mathbf{R1_{ct}}(\mathbf{true})) \wedge \\
\neg\,((Chaos_f^f \wedge ((P_f \wedge \frown/tr' = \frown/tr \wedge wait')\,;\, tr' - tr = \langle\langle\rangle\rangle))\,;\, \mathbf{R1_{ct}}(\mathbf{true})) \wedge \\
\neg\,(P_f^f \wedge ((Chaos_f \wedge \frown/tr' = \frown/tr \wedge wait')\,;\, head(tr' - tr) \neq \langle\rangle)) \wedge \\
\neg\,(Chaos_f^f \wedge ((P_f \wedge \frown/tr' = \frown/tr \wedge wait')\,;\, head(tr' - tr) \neq \langle\rangle)) \wedge \\
\neg\,((P_f^f \vee Chaos_f^f) \wedge head(diff(tr', tr)) \neq \langle\rangle) \\
\vdash \\
(P_f^t \wedge Chaos_f^t \wedge wait' \wedge \frown/tr' = \frown/tr) \vee (Diff(P_f^t, Chaos_f^t) \wedge (P_f^t \vee Chaos_f^t))
\end{array} \right) \\
& \qquad [\text{relational calculus}] \\
=& \mathbf{R_{ct}}(\mathbf{true}) \qquad [Chaos] \\
=& Chaos
\end{aligned}
$$

□

4.9 Parallel Composition

The parallel composition $P \,|[\, s_1 \mid \{\!|\, CS \,|\!\} \mid s_2 \,]|\, Q$ is the action where all events in the set CS must be synchronised, and the events outside CS can execute independently. In addition, s_1 and s_2 contain the local variables that each action can change during the composition. The parallel process terminates only if both P and Q terminate, and it becomes divergent if either of P and Q does so.

The parallel composition in the original *Circus Time* is constructed as the similar approach in the UTP. That is, it first transforms two actions into disjoint actions by renaming their alphabets, and generates the final result by a merge operation.

$$P \,|[\, s_1 \mid \{\!|\, CS \,|\!\} \mid s_2 \,]|\, Q \mathrel{\widehat{=}} (P\,;\, U0(out\alpha P) \wedge (Q\,;\, U1(out\alpha Q))_{+\{tr,ref\}}\,;\, M_{\parallel} \tag{4.71}$$

The labelling process $Ul(m)$ simply passes dashed variables of its predecessor to labelled variables and also removes these dashed variables from its alphabet, which is defined as

$$Ul(m) \mathrel{\widehat{=}} \mathbf{var}\ l.m\,;\, (l.m := m)\,;\, \mathbf{end}\ m$$

where **var** x and **end** x are the variable *declaration* and *undeclaration* respectively, and their definitions in the UTP book are given as

Definition 11. *Let A be an alphabet that includes x and x'. Then*

$$\mathbf{var}\ x \mathrel{\widehat{=}} \exists\, x \bullet \mathit{II}_A \qquad \alpha(\mathbf{var}\ x) = A \setminus \{x\}$$
$$\mathbf{end}\ x \mathrel{\widehat{=}} \exists\, x' \bullet \mathit{II}_A \qquad \alpha(\mathbf{end}\ x) = A \setminus \{x'\}$$

Notice that II_A is the relational identity ($\alpha \mathit{II}_A = A$). Therefore, through the variable declaration **var** and the variable undeclaration **end** operators, the output alphabet of $Ul(m)$ consists $l.m$ only. However under some circumstances we do need the initial values of P or Q. For this reason, we expand the alphabet after the labelling process. For example, $P_{+\{n\}}$ denotes $P \wedge n' = n$. Here we are only interested in tr and ref that will be used in $M_{\|}$.

The predicate $M_{\|}$ merges traces and refusals from its predecessor with respect to the interface CS, and also updates the variables from s_1 and s_2. The merge operation in the original *Circus Time* model is defined as

$$M_{\|} \mathrel{\widehat{=}} M_{ct}\,;\, \mathit{Skip} \tag{4.72}$$

$$M_{ct} \mathrel{\widehat{=}} \begin{pmatrix} ok' = (0.ok \wedge 1.ok) \wedge \mathit{MTR}(tr, tr', 0.tr, 1.tr, ref, ref', 0.ref, 1.ref) \wedge \\ wait' = (0.wait \vee 1.wait) \wedge state' = (0.state - s_2) \oplus (1.state - s_1) \end{pmatrix} \tag{4.73}$$

$$\mathit{MTR}\begin{pmatrix} tr, tr', 0.tr, 1.tr, \\ ref, ref', 0.ref, 1.ref \end{pmatrix} \mathrel{\widehat{=}} \begin{pmatrix} \mathit{diff}(tr', tr) \in \mathit{TSync}(\mathit{diff}(0.tr, tr), \mathit{diff}(1.tr, tr), CS) \wedge \\ ref' - \mathit{front}(ref) = \mathit{MRef}(0.ref - \mathit{front}(ref), 1.ref - \mathit{front}(ref), CS) \end{pmatrix} \tag{4.74}$$

where the predicate MTR is the merge operation for the traces and refusals. As shown in (4.74), the resulting trace, $\mathit{diff}(tr', tr)$, is a member of the set of traces generated by the synchronisation function TSync, and the corresponding refusals, $ref' - \mathit{front}(ref)$, is calculated by the function MRef on each time unit. The detailed definitions of TSync and MRef are given in Appendix C.

The parallel composition terminates only if both of them have terminated ($wait' = (0.wait \vee 1.wait)$), and becomes divergent only if one of them does so ($ok' = (0.ok \wedge 1.ok)$). In case M_{ct} may become divergent, the action Skip is used to capture the traces and refusals before the divergence. However, the definition (4.72) and its reactive design calculated from this definition cannot hold another of useful laws, $P \parallel \mathit{Chaos} = \mathit{Chaos}$. The CSP theory in the UTP uses the following predicate to hold this law,

$$\exists\, u((u \downarrow \alpha P = 0.tr - tr) \wedge (u \downarrow \alpha Q = 1.tr - tr) \wedge (u \downarrow \alpha(P \parallel Q) = u) \wedge tr' = tr \frown u) \tag{4.75}$$

where $s \downarrow E$ restricts a sequence s to only elements of the set E. However, the original *Circus Time* theory adopts only the predicate MTR that cannot retain the same result of (4.75). For example, the least trace of Chaos is $tr' = tr$, or $\mathit{diff}(tr', tr) = \langle\langle\rangle\rangle$. Suppose that the trace from another action is $\langle\langle a \rangle\rangle$. Through the predicate MTR, we may observe the trace $\langle\langle a \rangle\rangle$ rather than $\langle\langle\rangle\rangle$ that is what we really expect. To rule out this unwanted case, we give an extra predicate to tackle it in the precondition.

$$\neg\, (((P_f^f \vee Q_f^f) \wedge tr' = tr)\,;\, \mathbf{R1_{ct}}(\mathbf{true})) \tag{4.76}$$

This predicate states that, if either P or Q behaves like Chaos, the other will not be considered. Thus, we put the predicate (4.76) into the precondition of the reactive design (whose calculation can be found in Appendix C) that is calculated from the original definition (4.71), and finally obtain the following theorem.

Theorem 16. *(***Parallelism***)*

$$P \,|[\, s_1 \mid \{\!|\, CS \,|\!\}\mid s_2 \,]|\, Q =$$
$$\mathbf{R_{ct}}\left(\begin{array}{c} \neg\, (((P_f^f \vee Q_f^f) \wedge tr' = tr)\,;\, \mathbf{R1_{ct}}(\mathbf{true})) \wedge \\ \neg \left(\left(\begin{array}{l} \exists\, 0.tr, 0.ref, 1.tr, 1.ref \bullet \\ P_f^f[0.tr, 0.ref/tr', ref'] \wedge Q_f[1.tr, 1.ref/tr', ref'] \\ \wedge\ \mathit{MTR}(tr, tr', 0.tr, 1.tr, ref, ref', 0.ref, 1.ref) \end{array}\right)\,;\, \mathbf{R1_{ct}}(\mathbf{true})\right) \wedge \\ \neg \left(\left(\begin{array}{l} \exists\, 0.tr, 0.ref, 1.tr, 1.ref \bullet \\ P_f[0.tr, 0.ref/tr', ref'] \wedge Q_f^f[1.tr, 1.ref/tr', ref'] \\ \wedge\ \mathit{MTR}(tr, tr', 0.tr, 1.tr, ref, ref', 0.ref, 1.ref) \end{array}\right)\,;\, \mathbf{R1_{ct}}(\mathbf{true})\right) \\ \vdash \\ (((P_f^t\,;\, U0(out\alpha P)) \wedge (Q_f^t\,;\, U1(out\alpha Q)))_{+\{tr,ref\}};\ M_{ct}) \end{array}\right)$$

Note that if P or Q diverges eventually, we consider only the traces and their corresponding refusals that satisfy MTR, which is given in the precondition of the above reactive design. This reactive design cannot refine the definition in the original theory. However, we have proved a number of laws for parallelism that can validate this reactive design.

Law 64. $P \,|[\,|\, \{\!|\, CS \,|\!\}\, |\,]|\, Chaos = Chaos$

Proof.

$$P \,|[\,|\, \{\!|\, CS \,|\!\}\, |\,]|\, Chaos \qquad \text{[Theorem 16]}$$

$$= \mathbf{R_{ct}} \left(\begin{array}{c} \neg\, (((P_f^f \vee Chaos_f^f) \wedge tr' = tr)\,;\, \mathbf{R1_{ct}}(\mathbf{true})) \wedge \\ \neg \left(\left(\begin{array}{l} \exists\, 0.tr, 0.ref, 1.tr, 1.ref \bullet \\ \left(\begin{array}{l} P_f^f[0.tr, 0.ref/tr', ref'] \wedge Chaos_f[1.tr, 1.ref/tr', ref'] \\ \wedge\, MTR(tr, tr', 0.tr, 1.tr, ref, ref', 0.ref, 1.ref) \end{array} \right) \end{array} \right)\,;\, \mathbf{R1_{ct}}(\mathbf{true}) \right) \wedge \\ \neg \left(\left(\begin{array}{l} \exists\, 0.tr, 0.ref, 1.tr, 1.ref \bullet \\ \left(\begin{array}{l} P_f[0.tr, 0.ref/tr', ref'] \wedge Chaos_f^f[1.tr, 1.ref/tr', ref'] \\ \wedge\, MTR(tr, tr', 0.tr, 1.tr, ref, ref', 0.ref, 1.ref) \end{array} \right) \end{array} \right)\,;\, \mathbf{R1_{ct}}(\mathbf{true}) \right) \\ \vdash \\ (((P_f^t\,;\, U0(out\alpha P)) \wedge (Chaos_f^t\,;\, U1(out\alpha Chaos)))_{+\{tr,ref\}};\ M_{ct}) \end{array} \right)$$

[*Chaos* (Theorem 4)]

$$= \mathbf{R_{ct}} \left(\begin{array}{c} \neg\, (((P_f^f \vee \mathbf{true}) \wedge tr' = tr)\,;\, \mathbf{R1_{ct}}(\mathbf{true})) \wedge \\ \neg \left(\left(\begin{array}{l} \exists\, 0.tr, 0.ref, 1.tr, 1.ref \bullet \\ \left(\begin{array}{l} P_f^f[0.tr, 0.ref/tr', ref'] \wedge Chaos_f[1.tr, 1.ref/tr', ref'] \\ \wedge\, MTR(tr, tr', 0.tr, 1.tr, ref, ref', 0.ref, 1.ref) \end{array} \right) \end{array} \right)\,;\, \mathbf{R1_{ct}}(\mathbf{true}) \right) \wedge \\ \neg \left(\left(\begin{array}{l} \exists\, 0.tr, 0.ref, 1.tr, 1.ref \bullet \\ \left(\begin{array}{l} P_f[0.tr, 0.ref/tr', ref'] \wedge Chaos_f^f[1.tr, 1.ref/tr', ref'] \\ \wedge\, MTR(tr, tr', 0.tr, 1.tr, ref, ref', 0.ref, 1.ref) \end{array} \right) \end{array} \right)\,;\, \mathbf{R1_{ct}}(\mathbf{true}) \right) \\ \vdash \\ (((P_f^t\,;\, U0(out\alpha P)) \wedge (Chaos_f^t\,;\, U1(out\alpha Chaos)))_{+\{tr,ref\}};\ M_{ct}) \end{array} \right)$$

[relational calculus]

$$= \mathbf{R_{ct}} \left(\begin{array}{c} \neg\, ((P_f^f \wedge tr' = tr)\,;\, \mathbf{R1_{ct}}(\mathbf{true})) \wedge \neg\, \mathbf{true} \\ \neg \left(\left(\begin{array}{l} \exists\, 0.tr, 0.ref, 1.tr, 1.ref \bullet \\ \left(\begin{array}{l} P_f^f[0.tr, 0.ref/tr', ref'] \wedge Chaos_f[1.tr, 1.ref/tr', ref'] \\ \wedge\, MTR(tr, tr', 0.tr, 1.tr, ref, ref', 0.ref, 1.ref) \end{array} \right) \end{array} \right)\,;\, \mathbf{R1_{ct}}(\mathbf{true}) \right) \wedge \\ \neg \left(\left(\begin{array}{l} \exists\, 0.tr, 0.ref, 1.tr, 1.ref \bullet \\ \left(\begin{array}{l} P_f[0.tr, 0.ref/tr', ref'] \wedge Chaos_f^f[1.tr, 1.ref/tr', ref'] \\ \wedge\, MTR(tr, tr', 0.tr, 1.tr, ref, ref', 0.ref, 1.ref) \end{array} \right) \end{array} \right)\,;\, \mathbf{R1_{ct}}(\mathbf{true}) \right) \\ \vdash \\ (((P_f^t\,;\, U0(out\alpha P)) \wedge (Chaos_f^t\,;\, U1(out\alpha Chaos)))_{+\{tr,ref\}};\ M_{ct}) \end{array} \right)$$

[design and propositional calculus]

$$= \mathbf{R_{ct}}(\mathbf{true})$$

□

Law 65. $(a \rightarrow Skip) \,|||\, Miracle = Miracle$

Proof.

$$
\begin{array}{ll}
& (a \rightarrow Skip) \,|||\, Miracle & [\text{Def-}|||] \\
= & (\rightarrow Skip) \,|[\,|\, \{|\, \emptyset \,|\} \,|\,]|\, Miracle & \\
= & \mathbf{R_{ct}} \left(\begin{array}{c} \neg\, ((((a \rightarrow Skip)^f_f \vee Miracle^f_f) \wedge tr' = tr)\,;\, \mathbf{R1_{ct}}(\mathbf{true})) \wedge \\ \neg \left(\left(\begin{array}{l} \exists\, 0.tr, 0.ref, 1.tr, 1.ref \bullet \\ \left(\begin{array}{l} (a \rightarrow Skip)^f_f[0.tr, 0.ref/tr', ref'] \wedge Miracle_f[1.tr, 1.ref/tr', ref'] \\ \wedge\, MTR(tr, tr', 0.tr, 1.tr, ref, ref', 0.ref, 1.ref) \end{array} \right) \end{array} \right) ; \mathbf{R1_{ct}}(\mathbf{true}) \right) \wedge \\ \neg \left(\left(\begin{array}{l} \exists\, 0.tr, 0.ref, 1.tr, 1.ref \bullet \\ \left(\begin{array}{l} (a \rightarrow Skip)_f[0.tr, 0.ref/tr', ref'] \wedge Miracle^f_f[1.tr, 1.ref/tr', ref'] \\ \wedge\, MTR(tr, tr', 0.tr, 1.tr, ref, ref', 0.ref, 1.ref) \end{array} \right) \end{array} \right) ; \mathbf{R1_{ct}}(\mathbf{true}) \right) \\ \vdash \\ ((((a \rightarrow Skip)^t_f\,;\, U0(out\alpha(a \rightarrow Skip))) \wedge (Miracle^t_f\,;\, U1(out\alpha Miracle)))_{+\{tr,ref\}};\ M_{ct}) \end{array} \right) & \\
& & [Miracle \text{ (Theorem 5)}] \\
= & \mathbf{R_{ct}} \left(\begin{array}{c} \neg\, ((((a \rightarrow Skip)^f_f \vee \mathbf{false}) \wedge tr' = tr)\,;\, \mathbf{R1_{ct}}(\mathbf{true})) \wedge \\ \neg \left(\left(\begin{array}{l} \exists\, 0.tr, 0.ref, 1.tr, 1.ref \bullet \\ \left(\begin{array}{l} (a \rightarrow Skip)^f_f[0.tr, 0.ref/tr', ref'] \wedge \mathbf{false}[1.tr, 1.ref/tr', ref'] \\ \wedge\, MTR(tr, tr', 0.tr, 1.tr, ref, ref', 0.ref, 1.ref) \end{array} \right) \end{array} \right) ; \mathbf{R1_{ct}}(\mathbf{true}) \right) \wedge \\ \neg \left(\left(\begin{array}{l} \exists\, 0.tr, 0.ref, 1.tr, 1.ref \bullet \\ \left(\begin{array}{l} (a \rightarrow Skip)_f[0.tr, 0.ref/tr', ref'] \wedge \mathbf{false}[1.tr, 1.ref/tr', ref'] \\ \wedge\, MTR(tr, tr', 0.tr, 1.tr, ref, ref', 0.ref, 1.ref) \end{array} \right) \end{array} \right) ; \mathbf{R1_{ct}}(\mathbf{true}) \right) \\ \vdash \\ ((((a \rightarrow Skip)^t_f\,;\, U0(out\alpha(a \rightarrow Skip))) \wedge (\mathbf{false}\,;\, U1(out\alpha Miracle)))_{+\{tr,ref\}};\ M_{ct}) \end{array} \right) & \\
& & [\text{relational calculus}] \\
= & \mathbf{R_{ct}} \left(\begin{array}{c} \neg\, (((a \rightarrow Skip)^f_f \wedge tr' = tr); \mathbf{R1_{ct}}(\mathbf{true})) \wedge \neg\, \mathbf{false} \wedge \neg\, \mathbf{false} \wedge \neg\, \mathbf{false} \\ \vdash \\ \mathbf{false} \end{array} \right) & [\text{prop. calculus}] \\
= & \mathbf{R_{ct}}(\neg\, (((a \rightarrow Skip)^f_f \wedge tr' = tr); \mathbf{R1_{ct}}(\mathbf{true})) \vdash \mathbf{false}) & [\text{Simple prefix (Definition 10)}] \\
= & \mathbf{R_{ct}}(\neg\, ((\mathbf{false} \wedge tr' = tr); \mathbf{R1_{ct}}(\mathbf{true})) \vdash \mathbf{false}) & [\text{relational calculus}] \\
= & \mathbf{R_{ct}}(\mathbf{true} \vdash \mathbf{false}) & [Miracle] \\
= & Miracle &
\end{array}
$$

□

4.10 Hiding

Similar to the CSP hiding operator in the UTP [10], the hiding operator in the original *Circus Time* is defined as follows (Definition 9)

$$P \setminus CS \mathrel{\widehat{=}} \mathbf{R_{ct}}(\exists\, s, r \bullet P[s, r/tr', ref'] \wedge L_t)\,;\, Skip \tag{4.77}$$

$$L_t \mathrel{\widehat{=}} diff(tr', tr) = diff(s, tr) \downarrow_t (\Sigma - CS) \wedge r - front(ref) = ((ref' - front(ref)) \cup_t CS) \tag{4.78}$$

where two special operators, $\downarrow_t$ and $\cup_t$, are defined to restrict timed traces and complement refusals respectively.

$$tr_1 = (tr_2 \downarrow_t CS) \Leftrightarrow \forall\, i : 1..\#tr_1 \bullet tr_1(i) = (tr_2(i) \downarrow CS) \wedge \#tr_1 = \#tr_2$$
$$ref_1 = (ref_2 \cup_t CS) \Leftrightarrow \forall\, i : 1..\#ref_1 \bullet ref_1(i) = (ref_2(i) \cup CS) \wedge \#ref_1 = \#ref_2$$

As usual, the reactive design of the hiding operator is calculated as follows.

Lemma 24. *Providing P is a Circus Time action,*

$$\mathbf{R1_{ct}} \circ \mathbf{R2_{ct}}(\exists\, s, r \bullet P[s, r/tr', ref'] \wedge L_t) = \exists\, s, r \bullet P[s, r/tr', ref'] \wedge L_t$$

Proof.

TObedone.

□

Law 66. *Suppose P is a Circus Time action,*

$$(P \setminus CS)^f_f = \mathbf{R1_{ct}} \circ \mathbf{R2_{ct}}(\exists\, s, r \bullet P_f[s, r/tr', ref'] \land L_t)\,;\, \mathbf{R1_{ct}}(\mathbf{true})$$

Proof.

$$\begin{array}{ll}
& (P \setminus CS)^f_f & [\text{def of } \setminus] \\
=& (\mathbf{R_{ct}}(\exists\, s, r \bullet P[s, r/tr', ref'] \land L_t)\,;\, Skip)^f_f & [\text{relational calculus}] \\
=& (\mathbf{R_{ct}}(\exists\, s, r \bullet P[s, r/tr', ref'] \land L_t))_f\,;\, Skip^f & [\text{Lemma 43}] \\
=& (\mathbf{R_{ct}}(\exists\, s, r \bullet P[s, r/tr', ref'] \land L_t))_f\,;\, \mathbf{R1_{ct}}(\neg\, ok) & [\text{relational calculus}] \\
=& (\mathbf{R_{ct}}(\exists\, s, r \bullet P[s, r/tr', ref'] \land L_t))^f_f\,;\, \mathbf{R1_{ct}}(\mathbf{true}) & [\text{Law 24}] \\
=& (\mathbf{R1_{ct}} \circ \mathbf{R2_{ct}}(\exists\, s, r \bullet P_f[s, r/tr', ref'] \land (L_t)_f))^f\,;\, \mathbf{R1_{ct}}(\mathbf{true}) & [\text{substitution}] \\
=& \mathbf{R1_{ct}} \circ \mathbf{R2_{ct}}(\exists\, s, r \bullet P^f_f[s, r/tr', ref'] \land L_t)\,;\, \mathbf{R1_{ct}}(\mathbf{true}) & [\text{Law } \mathbf{??}] \\
=& (\exists\, s, r \bullet P^f_f[s, r/tr', ref'] \land L_t)\,;\, \mathbf{R1_{ct}}(\mathbf{true})
\end{array}$$

□

Law 67. *Suppose P is a Circus Time action,*

$$(P \setminus E)^t_f = ((\exists\, s, r \bullet P^f_f[s, r/tr', ref'] \land L_t)\,;\, \mathbf{R1_{ct}}(\mathbf{true}) \lor (\exists\, s, r \bullet P^t_f[s, r/tr', ref'] \land L_t))$$

Proof.

$$\begin{array}{ll}
& (P \setminus E)^t_f & [\text{def of } \setminus] \\
=& (\mathbf{R_{ct}}(\exists\, s, r \bullet P[s, r/tr', ref'] \land L_t)\,;\, Skip)^t_f & [\text{relational calculus}] \\
=& (\mathbf{R_{ct}}(\exists\, s, r \bullet P[s, r/tr', ref'] \land L_t))_f\,;\, Skip^t & [\text{Law } \mathbf{??}] \\
=& (\mathbf{R_{ct}}(\exists\, s, r \bullet P[s, r/tr', ref'] \land L_t))_f\,;\, (\mathbf{R1_{ct}}(\neg\, ok) \lor (ok \land \mathbb{I})) & [\text{rel. cal.}] \\
=& (\mathbf{R_{ct}}(\exists\, s, r \bullet P[s, r/tr', ref'] \land L_t))_f\,;\, \mathbf{R1_{ct}}(\neg\, ok) \lor & [\text{Step 4 in Law 66}] \\
& (\mathbf{R_{ct}}(\exists\, s, r \bullet P[s, r/tr', ref'] \land L_t))_f\,;\, (ok \land \mathbb{I}) & \\
=& (\exists\, s, r \bullet P^f_f[s, r/tr', ref'] \land L_t)\,;\, \mathbf{R1_{ct}}(\mathbf{true}) \lor & [\text{rel. calculus and unit law}] \\
& (\mathbf{R_{ct}}(\exists\, s, r \bullet P[s, r/tr', ref'] \land L_t))_f\,;\, (ok \land \mathbb{I}) & \\
=& (\exists\, s, r \bullet P^f_f[s, r/tr', ref'] \land L_t)\,;\, \mathbf{R1_{ct}}(\mathbf{true}) \lor (\exists\, s, r \bullet P[s, r/tr', ref'] \land L_t)^t_f & [\text{substitution}] \\
=& (\exists\, s, r \bullet P^f_f[s, r/tr', ref'] \land L_t)\,;\, \mathbf{R1_{ct}}(\mathbf{true}) \lor (\exists\, s, r \bullet P^t_f[s, r/tr', ref'] \land L_t)
\end{array}$$

□

Now, the reactive design of hiding can be deduced in terms of the above laws.

Theorem 17. *(**Hiding**) Suppose P is a Circus Time action,*

$$P \setminus CS = \mathbf{R_{ct}}(\neg\, ((\exists\, s, r \bullet P^f_f[s, r/tr', ref'] \land L_t)\,;\, \mathbf{R1_{ct}}(\mathbf{true})) \vdash (\exists\, s, r \bullet P^t_f[s, r/tr', ref'] \land L_t))$$

Proof.

$$
\begin{aligned}
& P \setminus CS && \text{[Theorem 1]}\\
&= \mathbf{R_{ct}}(\neg\, (P \setminus CS)^f_f \vdash (P \setminus CS)^t_f) && \text{[def of design]}\\
&= \mathbf{R_{ct}}(ok \wedge \neg\, (P \setminus CS)^f_f \Rightarrow ok' \wedge (P \setminus CS)^t_f) && \text{[Law 66 and Law 67]}\\
&= \mathbf{R_{ct}}\left(\begin{array}{l} ok \wedge \neg\, ((\exists\, s, r \bullet P^f_f[s, r/tr', ref'] \wedge L_t)\,;\, \mathbf{R1_{ct}}(\mathbf{true})) \Rightarrow \\ ok' \wedge \left(\begin{array}{c} (\exists\, s, r \bullet P^f_f[s, r/tr', ref'] \wedge L_t)\,;\, \mathbf{R1_{ct}}(\mathbf{true}) \\ \vee\, (\exists\, s, r \bullet P^t_f[s, r/tr', ref'] \wedge L_t) \end{array}\right) \end{array}\right) && \text{[prop. cal.]}\\
&= \mathbf{R_{ct}}(ok \wedge \neg\, ((\exists\, s, r \bullet P^f_f[s, r/tr', ref'] \wedge L_t);\, \mathbf{R1_{ct}}(\mathbf{true})) \Rightarrow ok' \wedge (\exists\, s, r \bullet P^t_f[s, r/tr', ref'] \wedge L_t)) && \text{[design]}\\
&= \mathbf{R_{ct}}(\neg\, ((\exists\, s, r \bullet P^f_f[s, r/tr', ref'] \wedge L_t)\,;\, \mathbf{R1_{ct}}(\mathbf{true})) \vdash (\exists\, s, r \bullet P^t_f[s, r/tr', ref'] \wedge L_t))
\end{aligned}
$$

□

4.11 Recursion

The semantics of recursion is the same as that in the UTP [10]: weakest fixed point. Given a monotonic function F, the semantics of recursion is the weakest fixed point of F.

$$\mu\, X \bullet F(X) \mathrel{\widehat{=}} \bigsqcap \{X \mid F(X) \sqsubseteq X\} \tag{4.79}$$

The strongest fixed point of $F(X)$ is defined as the dual of the weakest.

$$\nu F \mathrel{\widehat{=}} \neg\ \mu\, X \bullet \neg\, F(\neg\, X) \tag{4.80}$$

To express a recursion as a reactive design, we have to calculate the precondition and postcondition of a recursively defined design. For that, we can use the definition of a recursive design and some theorems on linking theories in [10]. In the theory of designs, any monotonic function of designs can be expressed in terms of a pair of function that apply separately to the precondition and the postcondition, for example

$$F(P, Q) \vdash G(P, Q)$$

Here, P and Q are predicates representing the precondition and postcondition of a design, F is monotonic in P and antimonotonic in Q, whereas G is monotonic in Q and antimonotonic in P. Thus, as described in the theory of designs, the weakest fixed point is given by a mutually recursive formula, that we reproduce below.

Law 68. *(Property 1-L4)*

$$
\begin{aligned}
& \mu(X, Y) \bullet (F(X, Y) \vdash G(X, Y)) = P(Q) \vdash Q\\
\text{where}\quad & P(Y) = \nu X \bullet F(X, Y)\\
\text{and}\quad & Q = \mu\, Y \bullet (P(Y) \Rightarrow G(P(Y), Y))
\end{aligned}
$$

As shown in Theorem 1, if X is a reactive design, $X = \mathbf{R_{ct}}(\neg\, X^f_f \vdash X^t_f)$. Hence, similar to Law 68, the weakest fixed point of a recursively reactive design can also be given by a mutually recursive formula.

Theorem 18. *(***Recursion***)*

$$\mu(X, Y) \bullet \mathbf{R_{ct}}(F(X, Y) \vdash G(X, Y)) = \mathbf{R_{ct}}(\mu(X, Y) \bullet F(X, Y) \vdash G(X, Y))$$

Clearly, this theorem states that the left part of the equation is the natural expression of a recursively reactive design and the right part is the solution to calculate it. That is, the weakest fixed point of the recursively reactive design is $\mathbf{R_{ct}}$ healthiness of the weakest fixed point of its recursive design. The calculation of $\mu(X, Y) \bullet F(X, Y) \vdash G(X, Y)$ is obviously by Law 68.

To prove the equation in Theorem 18, we directly adopt an important theorem from the linking theories of the UTP book, which can be described here.

Theorem 19. *Let D and E be monotonic functions. If there exists a function R such that $R \circ D = E \circ R$, then $R(\mu\, D) = \mu\, E$.*

As a result, the proof of Theorem 18 can be established as follows.

Proof.

$$\begin{array}{ll} \text{Let} & D(X \vdash Y) = F(X, Y) \vdash G(X, Y) \ \ \textit{and} \\ & E(\mathbf{R_{ct}}(X \vdash Y)) = \mathbf{R_{ct}}(F(X, Y) \vdash G(X, Y)) \end{array}$$

$$\begin{array}{lll} \text{then} & E \circ \mathbf{R_{ct}}(X \vdash Y) & [\text{def of } E] \\ & = \mathbf{R_{ct}}(F(X, Y) \vdash G(X, Y)) & [\text{def of } D] \\ & = \mathbf{R_{ct}}(D(X \vdash Y) & [\text{def of composition}] \\ & = \mathbf{R_{ct}} \circ D(X \vdash Y) & \end{array}$$

$$\text{therefore} \quad \mu(X, Y) \bullet E(\mathbf{R_{ct}}(X \vdash Y)) = \mathbf{R_{ct}}(\mu(X, Y) \bullet D(X \vdash Y))$$

□

4.12 Timed Event Prefix

A timed event prefix $c.e@t \rightarrow P$ allows the action to record the amount of time which has elapsed between the initial event's offer and its occurrence. The information of time may then be used by the subsequent action as a local variable. Similar to the composition of an untimed event prefix in Section 4.6, $c.e@t \rightarrow P$ can also be expressed as $c.e@t \rightarrow \mathit{Skip} \,;\, P$. Thus, we only need to give the definition of a simple timed event prefix.

Theorem 20.

$$c.e@t \rightarrow \mathit{Skip} \mathrel{\widehat{=}} \mathbf{R_{ct}} \left(\mathbf{true} \vdash \left(\begin{array}{l} \mathit{wait\_com\_time}(c) \lor \mathit{term\_now\_com\_time}(c.e) \lor \\ \mathit{terminating\_com\_time}(c.e, t) \end{array} \right) \right)$$

$$\mathit{wait\_com\_time}(c) \mathrel{\widehat{=}} (\mathit{wait}' \land \mathit{state}' = \mathit{state} \oplus \{t \mapsto \#tr' - \#tr\} \land \mathit{possible}(\mathit{ref}, \mathit{ref}', c) \land {}^\frown/tr' = {}^\frown/tr)$$

$$\mathit{term\_now\_com\_time}(c.e) \mathrel{\widehat{=}} \left(\begin{array}{l} \neg\ \mathit{wait}' \land \mathit{state}' = \mathit{state} \oplus \{t \mapsto \#tr' - \#tr\} \land \\ \mathit{diff}(tr', tr) = \langle\langle c.e \rangle\rangle \land \mathit{front}(\mathit{ref}') = \mathit{front}(\mathit{ref}) \end{array} \right)$$

$$\mathit{term\_next\_com\_time}(c.e) \mathrel{\widehat{=}} \left(\begin{array}{c} \neg\ \mathit{wait}' \land \mathit{state}' = \mathit{state} \oplus \{t \mapsto \#tr' - \#tr\} \\ \land\ tr' - tr = \langle\langle c.e \rangle\rangle \land \mathit{front}(\mathit{ref}') = \mathit{ref} \end{array} \right)$$

$$\mathit{terminating\_com\_time}(c.e) \mathrel{\widehat{=}} (\mathit{wait\_com}(c) \,;\, \mathit{term\_next\_com\_com}(c.e))$$

These predicates within the above definition are very similar to those in the definition of prefix in Section 4.6.

4.13 Timeout

The timeout operator in the original *Circus Time* is the same as that in Timed CSP, which takes advantage of the urgency of internal events caused by the hiding operator. That is, Q will take place if no observable event in P can happen within d time units.

$$P \rhd \{d\}\ Q = (P \,\Box\, (\mathit{Wait}\ d \,;\, c \rightarrow Q)) \setminus \{c\} \qquad (c \notin \alpha P \cup \alpha Q)$$

However, the reactive design of the timeout operator is very complex and nearly unreadable if we begin it with the reactive designs of *Wait*, sequential composition, external choice and hiding, in light of the definition above.

We intuitively give a simpler reactive design which includes the whole scenario of the timeout operator.

Theorem 21.

$$P \rhd \{d\}\ Q \mathrel{\widehat{=}} \mathbf{R_{ct}}\left(\begin{array}{c} \neg\,((P_f^f \wedge {}^\frown\!/tr' = {}^\frown\!/tr \wedge \#tr' - \#tr \leq d)\,;\, \mathbf{R1_{ct}}(\mathbf{true})) \wedge \\ \neg\,(P_f^f \wedge (({}^\frown\!/tr' = {}^\frown\!/tr \wedge \#tr' - \#tr < d \wedge wait')\,;\, head(tr' - tr) \neq \langle\rangle)) \wedge \\ \neg\,(P_f^f \wedge head(diff\,(tr, tr)) \neq \langle\rangle) \wedge \\ \neg\,((P_f^t \wedge {}^\frown\!/tr' = {}^\frown\!/tr \wedge \#tr' - \#tr = d)\,;\, (\mathit{II}^{-wait} \wedge \neg\, wait')\,;\, Q_f^f) \\ \vdash \\ (P_f^t \wedge wait' \wedge {}^\frown\!/tr' = {}^\frown\!/tr \wedge \#tr' - \#tr < d) \vee \\ (P_f^t \wedge (((^\frown\!/tr' = {}^\frown\!/tr \wedge \#tr' - \#tr < d \wedge wait')\,;\, term\_next) \vee term\_now)) \vee \\ ((P_f^t \wedge ({}^\frown\!/tr' = {}^\frown\!/tr \wedge \#tr' - \#tr = d))\,;\, (\mathit{II}^{-wait} \wedge \neg\, wait')\,;\, Q_f^t) \end{array}\right)$$

From the above definition, we have a clear view of the behaviour of the timeout operator. In the precondition, the first clause in the conjunction states that P diverges before d time units without executing any external event, including the immediate divergence of P; the second states that P has waited for a whole and executed some external events before d but becomes divergent later; the third states that P executes the external events immediately and diverges later; the fourth states that Q diverges if P has not executed any observable event before d. The postcondition describes three possible cases. First, P can wait for the interaction of its environment less than d time units. Second, P can terminate or execute external events within d. Finally, Q takes place if nothing happens in P by the end of d time units. Note that we use the predicate $(\mathit{II}^{-wait} \wedge \neg\, wait')$ and sequential composition to guarantee that Q starts immediately after d while the final state of P at d is passed on as the initial state of Q.

The delay operator *Wait* can also be defined by the timeout operator, *Stop* and *Skip* as

$$Wait\ d \mathrel{\widehat{=}} Stop \rhd \{d\} Skip$$

Since the reactive designs of *Skip* and *Stop* have been given in Section 4.1 and 4.4, we, here, treat this alternative definition of *Wait* as a law and furthermore prove it to show the consistency of this reactive design of timeout.

Law 69. $Wait\ d \mathrel{\widehat{=}} Stop \rhd \{d\} Skip$

Proof.

$$\begin{array}{ll} & Stop \rhd \{d\} Skip \qquad \text{[Theorem 21 (timeout)]} \\ = \mathbf{R_{ct}}\left(\begin{array}{c} \neg\,(((Stop)_f^f \wedge {}^\frown\!/tr' = {}^\frown\!/tr \wedge \#tr' - \#tr \leq d)\,;\, \mathbf{R1_{ct}}(\mathbf{true})) \wedge \\ \neg\,((Stop)_f^f \wedge (({}^\frown\!/tr' = {}^\frown\!/tr \wedge \#tr' - \#tr < d \wedge wait')\,;\, head(tr' - tr) \neq \langle\rangle)) \wedge \\ \neg\,((Stop)_f^f \wedge head(diff\,(tr, tr)) \neq \langle\rangle) \wedge \\ \neg\,(((Stop)_f^t \wedge {}^\frown\!/tr' = {}^\frown\!/tr \wedge \#tr' - \#tr = d)\,;\, (\mathit{II}^{-wait} \wedge \neg\, wait')\,;\, (Skip)_f^f) \\ \vdash \\ ((Stop)_f^t \wedge wait' \wedge {}^\frown\!/tr' = {}^\frown\!/tr \wedge \#tr' - \#tr < d) \vee \\ ((Stop)_f^t \wedge (((^\frown\!/tr' = {}^\frown\!/tr \wedge \#tr' - \#tr < d \wedge wait')\,;\, term\_next) \vee term\_now)) \vee \\ (((Stop)_f^t \wedge ({}^\frown\!/tr' = {}^\frown\!/tr \wedge \#tr' - \#tr = d))\,;\, (\mathit{II}^{-wait} \wedge \neg\, wait')\,;\, (Skip)_f^t) \end{array}\right) \\ & \qquad \text{[}Stop\text{ and }Skip\text{ and their preconditions]} \\ = \mathbf{R_{ct}}\left(\begin{array}{c} \neg\,((\mathbf{false} \wedge {}^\frown\!/tr' = {}^\frown\!/tr \wedge \#tr' - \#tr \leq d)\,;\, \mathbf{R1_{ct}}(\mathbf{true})) \wedge \\ \neg\,(\mathbf{false} \wedge (({}^\frown\!/tr' = {}^\frown\!/tr \wedge \#tr' - \#tr < d \wedge wait')\,;\, head(tr' - tr) \neq \langle\rangle)) \wedge \\ \neg\,(\mathbf{false} \wedge head(diff\,(tr, tr)) \neq \langle\rangle) \wedge \\ \neg\,(((Stop)_f^t \wedge {}^\frown\!/tr' = {}^\frown\!/tr \wedge \#tr' - \#tr = d)\,;\, (\mathit{II}^{-wait} \wedge \neg\, wait')\,;\, \mathbf{false}) \\ \vdash \\ ((Stop)_f^t \wedge wait' \wedge {}^\frown\!/tr' = {}^\frown\!/tr \wedge \#tr' - \#tr < d) \vee \\ ((Stop)_f^t \wedge (((^\frown\!/tr' = {}^\frown\!/tr \wedge \#tr' - \#tr < d \wedge wait')\,;\, term\_next) \vee term\_now)) \vee \\ (((Stop)_f^t \wedge ({}^\frown\!/tr' = {}^\frown\!/tr \wedge \#tr' - \#tr = d))\,;\, (\mathit{II}^{-wait} \wedge \neg\, wait')\,;\, (Skip)_f^t) \end{array}\right) \qquad \text{[rel. cal.]} \end{array}$$

$$= \mathbf{R_{ct}}\left(\mathbf{true} \vdash \left(\begin{array}{c}((Stop)^t_f \wedge wait' \wedge \frown/tr' = \frown/tr \wedge \#tr' - \#tr < d) \vee \\ ((Stop)^t_f \wedge (((\frown/tr' = \frown/tr \wedge \#tr' - \#tr < d \wedge wait')\,;\, term\_next) \vee term\_now)) \vee \\ (((Stop)^t_f \wedge (\frown/tr' = \frown/tr \wedge \#tr' - \#tr = d))\,;\, (\mathit{II}^{-wait} \wedge \neg\, wait')\,;\, (Skip)^t_f)\end{array}\right)\right)$$

[*Stop* contradicts with *term_now* and *term_next*]

$$= \mathbf{R_{ct}}\left(\mathbf{true} \vdash \left(\begin{array}{c}((Stop)^t_f \wedge wait' \wedge \frown/tr' = \frown/tr \wedge \#tr' - \#tr < d) \vee \\ (((Stop)^t_f \wedge (\frown/tr' = \frown/tr \wedge \#tr' - \#tr = d))\,;\, (\mathit{II}^{-wait} \wedge \neg\, wait')\,;\, (Skip)^t_f)\end{array}\right)\right)$$

[*Stop* and *Skip* and their postconditions]

$$= \mathbf{R_{ct}}\left(\mathbf{true} \vdash \left(\begin{array}{c}(wait' \wedge \frown/tr' = \frown/tr \wedge \#tr' - \#tr < d) \vee \\ ((\frown/tr' = \frown/tr \wedge \#tr' - \#tr = d)\,;\, (\mathit{II}^{-wait} \wedge \neg\, wait')\,;\, (wait' \wedge tr' = tr \wedge state' = state))\end{array}\right)\right)$$

[relational calculus]

$$= \mathbf{R_{ct}}(\mathbf{true} \vdash \frown/tr' = \frown/tr \wedge (\#tr' - \#tr < d \lhd wait' \rhd \#tr' - \#tr = d \wedge state' = state))$$

□

4.14 Deadline

The deadline operators in this new model are *hard*. That is, the deadlines *must* be satisfied. Failure to meet a deadline in *Circus Time* will result in an infeasible action. Here we define two kinds of deadlines: one (◀) is an action must execute an observable event within a deadline, the other (▶) is the action must terminate within the deadline.

Theorem 22.

$$P \blacktriangleright d \mathrel{\widehat{=}} \mathbf{R_{ct}}\left(\begin{array}{c}\neg\,((P^f_f \wedge tr' = tr)\,;\, \mathbf{R1_{ct}}(\mathbf{true})) \wedge \neg\,((P^f_f \wedge \#tr' - \#tr \leq d)\,;\, \mathbf{R1_{ct}}(\mathbf{true})) \\ \vdash \\ (P^t_f \wedge \#tr' - \#tr \leq d))\end{array}\right)$$

The above definition guarantees that only the observation within d time units is valid. In other words, if P cannot terminate within d time units, the time will stop at d. As a matter of fact, this deadline operator incurs a timelock or timestop, which prevents time from passing beyond a certain point, if the deadline cannot be satisfied.

The deadline operator (◀), which requests that external events must happen within d from the start, is defined by external choice, delay and *Miracle* as

$$d \blacktriangleleft P \mathrel{\widehat{=}} P \,\Box\, (Wait\ d\,;\, Miracle) \tag{4.81}$$

Note that, here, *Miracle* plays an important role to generate a timelock after d time units if the external choice cannot be resolved by P. The reactive design of this deadline operator is deduced from (4.81) and the detailed proof can be found in Appdendix D.

Theorem 23.

$$d \blacktriangleleft P = \mathbf{R_{ct}}\left(\begin{array}{c}\neg\,((P^f_f \wedge tr' = tr)\,;\, \mathbf{R1_{ct}}(\mathbf{true})) \wedge \\ \neg\,((P^f_f \wedge \#tr' - \#tr \leq d \wedge \frown/tr' = \frown/tr);\, \mathbf{R1_{ct}}(\mathbf{true})) \wedge \\ \neg\,(P^f_f \wedge ((\#tr' - \#tr < d \wedge \frown/tr' = \frown/tr \wedge wait')\,;\, head(tr' - tr) \neq \langle\rangle)) \wedge \\ \neg\,(P^f_f \wedge head(diff(tr', tr)) \neq \langle\rangle) \\ \vdash \\ (P^t_f \wedge \#tr' - \#tr < d \wedge wait' \wedge \frown/tr' = \frown/tr) \vee \\ (P^t_f \wedge (((\#tr' - \#tr < d \wedge \frown tr' = \frown tr \wedge wait');\, term\_next) \vee term\_now))\end{array}\right)$$

From the precondition of this reactive design, the first clause states that P diverges immediately; the second states that P diverges without executing any external event; the third expresses that P waits for a while, executes some external events and then diverges later; the fourth denotes that P immediately executes the external events and diverges afterwards. From the postcondition, it actually states that P must either terminate or execute certain observable events before or at d time units. Note that *term_now* and *term_next* have been defined in Section 4.8. Similar to ▶, fail to meet the deadline can result in a timelock.

4.15 Interrupts

Hoare's CSP book [9] gives a generic interrupt operator, $P \triangle Q$, which allows P to execute, but it may be interrupted by the first external event from Q and the program control is simultaneously passed to Q. Thereafter, the standard models of CSP adopt a catastrophic interrupt, which is expressed as $P \triangle_c Q$. Here, the catastrophic event c is unique, and its occurrence can interrupt P. However, this simpler interrupt might not be convenient for specifying real-time systems. For example, a seminar room is booked for an hour. Therefore one hour later after the punctual start, the seminar has to be interrupted if the next session has been booked by someone else. But the speaker may continue his talk if no one turns up to use this room. This example may be described as follows

$$SEMINAR \triangle (Wait\ 1\,;\ close \rightarrow Skip)$$

Timed CSP [18] uses the generic interrupt to satisfy time requirements. Note that the above process does not mean that the interrupt must occur after one hour because the occurrence of *close* depends on the environment. If we say that the seminar must finish after one hour no matter whether this room will be used then, the generic interrupt is incompetent. Accordingly, a timed interrupt is introduced in Timed CSP to describe this scenario.

$$SEMINAR \triangle_1 Skip$$

which means the interrupt must happen one hour later. That is to say, the timed interrupt is time-driven and out of control of its environment. The UTP semantics of the catastrophe in CSP has been discussed in [11], and we will use the same idea to calculate its reactive design in *Circus Time*. Unfortunately, the approach in [11] does not work for the generic interrupt. Therefore, we follow the idea in Timed CSP to deal with the generic interrupt to consider it a special kind of parallel composition. For the timed interrupt, it can still be treated as a sequential composition and therefore its reactive design is directly given.

4.15.1 Catastrophe

We use the same approach in [11] but with changes to accommodate time behaviours to generate a UTP definition for a catastrophic interrupt, which is then calculated to produce a reactive design. The general idea in [11] is to use a new healthiness condition **I3**, whose name simply reflects its relation to $\mathbf{R3_{ct}}$, to bring the catastrophic event forward to any waiting state of the interruptible process while this event is not refused by an alphabet extension. An **I3** healthy process can only execute while its predecessor is in an intermediate state, or can behave like a *Circus Time* identity if its predecessor terminates.

Definition 12. $\mathbf{I3}(P) = P \lhd wait \rhd \mathit{II}_{ct}$

Here we can clearly see **I3**'s relation to $\mathbf{R3_{ct}}$ by the law $\mathbf{R3_{ct}}(\mathbf{I3}(P)) = \mathit{II}_{ct}$ which states that an **I3** healthy process will behave as the identity if it is required to be $\mathbf{R3_{ct}}$ healthy. In addition, to make sure the interrupt event is not refused during the execution of the interruptible process, an alphabet extension operator is defined as

Definition 13. $P^{+c} \mathrel{\widehat=} (P \wedge possible(ref, ref', c)); \left(\mathit{II} \lhd wait \rhd \left(\begin{array}{c} \mathit{II}^{-ref} \wedge \\ front(ref') = front(ref) \end{array} \right) \right)$

where $possible(ref, ref', c)$ is defined in Section 4.6. The predicate II^{-ref} is the relational identity without the variables ref and ref'. Note that we use $front(ref') = front(ref)$ to free the last element of refusals in P, since we usually request that the last refusal is arbitrary if a process terminates. Furthermore, we develop a new predicate, $interrupt(c, Q)$, to describe that an event is forced to occur despite an apparent situation opposite to an ordinary action in *Circus Time*.

Definition 14.

$$try(c, Q) \mathrel{\widehat=} (\mathit{II} \lhd wait' \rhd term\_now\_com(c));\ Q$$

Note that $term\_now\_com(c)$ has been defined in Section 4.6. The action in $try(c, Q)$ is similar to the prefix operator in *Circus Time* but without the constraint of the healthiness conditions. Also, it simplifies the behaviour of the prefix so as to terminate with the immediate execution of $c.e$, or just behave like the identity. Here, the usual non-refusal of c will be achieved by the alphabet extension when sequentially composed with the interruptible action.

Definition 15. $force(c, Q) \mathrel{\widehat{=}} \mathbf{I3}(try(c, Q))$

The definition of $force(c, Q)$ in Definition 15 is an **I3**-healthy $try(c, Q)$ that states that it behaves as the identity ($\mathbb{I}_{ct}$) when its predecessor terminates, and otherwise behaves like $try(c, Q)$. Thus, the predicate $interrupt(c, Q)$ is defined as a $\mathbf{CSP1_{ct}}$-healthy *force*, which considers divergences of its predecessor and Q.

Definition 16. $interrupt(c, Q) \mathrel{\widehat{=}} \mathbf{CSP1_{ct}}(force(c, Q))$

The definition for catastrophe is given as a sequential composition between the interruptible action P with an alphabet extension by augmenting the interrupt event c, and the newly-defined predicate $interrupt(c, Q)$.

Definition 17. $P \triangle_c Q \mathrel{\widehat{=}} \mathbf{R3_{ct}} \circ \mathbf{CSP2_{ct}}(P^{+c}\,;\, interrupt(c, Q))$

Here, $\mathbf{R3_{ct}}$ restricts the bound of **I3**, and $\mathbf{CSP2_{ct}}$ requires that a divergence within this interrupt may also contain termination.

To obtain the reactive design definition of catastrophe, we calculate it as follows.

Lemma 25. $(P^{+c}\,;\, (\neg\, ok \wedge RT))_f = (P^f_f \wedge possible(ref, ref', c))\,;\, \mathbf{R1_{ct}}(\mathbf{true})$

Proof.

$$
\begin{aligned}
&(P^{+c}\,;\, (\neg\, ok \wedge RT))_f && \text{[Definition 13]}\\
=&((P \wedge possible(ref, ref', c))\,;\, \big(\mathbb{I} \lhd wait \rhd \big(\ \mathbb{I}^{-ref} \wedge front(ref') = front(ref)\ \big)\big)\,;\, (\neg\, ok \wedge RT))_f\\
& && \text{[relational calculus]}\\
=&(P_f \wedge possible(ref, ref', c))\,;\, \big(\mathbb{I} \lhd wait \rhd \big(\ \mathbb{I}^{-ref} \wedge front(ref') = front(ref)\ \big)\big)\,;\, (\neg\, ok \wedge RT)\\
& && \text{[relational calculus]}\\
=&(P_f \wedge possible(ref, ref', c))\,;\, (\mathbb{I} \wedge wait)\,;\, (\neg\, ok \wedge RT) \vee\\
&(P_f \wedge possible(ref, ref', c))\,;\, (\neg\, wait \wedge \mathbb{I}^{-ref} \wedge front(ref') = front(ref))\,;\, (\neg\, ok \wedge RT)\\
& && \text{[relational calculus]}\\
=&(P^f_f \wedge possible(ref, ref', c));\ RT \vee (P^f_f \wedge possible(ref, ref', c));\ RT && \text{[propositional calculus]}\\
=&(P^f_f \wedge possible(ref, ref', c));\ \mathbf{R1_{ct}}(\mathbf{true})
\end{aligned}
$$

□

Lemma 26. $P^{+c}\,;\, (ok \wedge try(c, Q) \wedge wait) =$

$$
\left(\begin{array}{l}((P \wedge possible(ref, ref', c) \wedge ok' \wedge wait') \vee\\ ((P \wedge possible(ref, ref', c))\,;\, (ok \wedge wait \wedge term\_now\_com(c))\,;\, (\neg\, wait \wedge Q_f)\end{array}\right)
$$

Proof.

$$
\begin{aligned}
&P^{+c}\,;\, (ok \wedge try(c, Q) \wedge wait) && \text{[Definition 14]}\\
=&P^{+c}\,;\, (ok \wedge ((\mathbb{I} \lhd wait' \rhd term\_now\_com(c));\ Q) \wedge wait) && \text{[Definition 13]}\\
=&\left(\begin{array}{l}(P \wedge possible(ref, ref', c))\,;\, \big(\mathbb{I} \lhd wait \rhd \big(\ \mathbb{I}^{-ref} \wedge front(ref') = front(ref)\ \big)\big)\ ;\\ (ok \wedge ((\mathbb{I} \lhd wait' \rhd term\_now\_com(c));\ Q) \wedge wait)\end{array}\right)\\
& && \text{[rel. calculus]}\\
=&\left(\begin{array}{l}(P \wedge possible(ref, ref', c))\,;\, (\mathbb{I} \wedge wait)\,;\, (ok \wedge wait \wedge ((\mathbb{I} \wedge wait');\ Q)) \vee\\ \left(\begin{array}{l}(P \wedge possible(ref, ref', c))\,;\, (\mathbb{I}^{-ref} \wedge wait \wedge front(ref') = front(ref))\,;\\ (ok \wedge wait \wedge ((\neg\, wait' \wedge term\_now\_com(c));\ Q))\end{array}\right)\end{array}\right)\\
& && \text{[rel. calculus and } front(ref') = front(ref) \text{ in } term\_now\_com(c)\text{]}\\
=&\left(\begin{array}{l}((P \wedge possible(ref, ref', c))\,;\, (ok \wedge wait \wedge \mathbb{I} \wedge wait');\ Q) \vee\\ \left(\ (P \wedge possible(ref, ref', c))\,;\, (ok \wedge wait \wedge \neg\, wait' \wedge term\_now\_com(c));\ Q\ \right)\end{array}\right) && [Q \text{ is } \mathbf{R3_{ct}}]\\
=&\left(\begin{array}{l}((P \wedge possible(ref, ref', c))\,;\, (ok \wedge wait \wedge \mathbb{I} \wedge wait');\ (wait \wedge \mathbb{I}_{ct})) \vee\\ \left(\ (P \wedge possible(ref, ref', c))\,;\, (ok \wedge wait \wedge \neg\, wait' \wedge term\_now\_com(c));\ (\neg\, wait \wedge Q_f)\ \right)\end{array}\right)\\
& && \text{[relational calculus]}\\
=&\left(\begin{array}{l}((P \wedge possible(ref, ref', c))\,;\, (ok \wedge wait \wedge \mathbb{I} \wedge wait');\ (wait \wedge ok' \wedge \mathbb{I})) \vee\\ \left(\ (P \wedge possible(ref, ref', c))\,;\, (ok \wedge wait \wedge term\_now\_com(c));\ (\neg\, wait \wedge Q_f)\ \right)\end{array}\right) && \text{[rel. calculus]}\\
=&\left(\begin{array}{l}((P \wedge possible(ref, ref', c) \wedge ok' \wedge wait') \vee\\ ((P \wedge possible(ref, ref', c))\,;\, (ok \wedge wait \wedge term\_now\_com(c))\,;\, (\neg\, wait \wedge Q_f)\end{array}\right)
\end{aligned}
$$

□

Lemma 27.

$$(P \triangle_i Q)^f_f = \left(\begin{array}{l} ((P^f_f \wedge \mathit{possible}(\mathit{ref}, \mathit{ref}', c)); \mathbf{R1_{ct}}(\mathbf{true})) \vee \\ ((P^t_f \wedge \mathit{possible}(\mathit{ref}, \mathit{ref}', c)); (\mathit{wait} \wedge \mathit{term\_now\_com}(c)); (\neg\ \mathit{wait} \wedge Q^f_f)) \end{array} \right)$$

Proof.

$$\begin{array}{ll}
& (P \triangle_{c.e} Q)^f_f & \text{[Def-17]} \\
= & (\mathbf{R3_{ct}} \circ \mathbf{CSP2_{ct}}(P^{+c.e} ; \mathit{interrupt}(c.e, Q)))^f_f & [\mathbf{R3_{ct}} \text{ and substitution}] \\
= & (\mathbf{CSP2_{ct}}(P^{+c.e} ; \mathit{interrupt}(c.e, Q)))^f_f & [\mathbf{CSP2_{ct}}] \\
= & ((P^{+c.e} ; \mathit{interrupt}(c.e, Q)) ; J)^f_f & \text{[J-split]} \\
= & ((P^{+c.e} ; \mathit{interrupt}(c.e, Q))^f \vee (ok' \wedge (P^{+c.e} ; \mathit{interrupt}(c.e, Q))^t))^f_f & \text{[subs.]} \\
= & (P^{+c.e} ; \mathit{interrupt}(c.e, Q))^f_f & \text{[Def-16]} \\
= & (P^{+c.e} ; \mathbf{CSP1_{ct}}(\mathit{force}(c.e, Q)))^f_f & [\mathbf{CSP1_{ct}}] \\
= & (P^{+c.e} ; ((\neg\ ok \wedge RT) \vee (ok \wedge \mathit{force}(c.e, Q))))^f_f & \text{[relational calculus]} \\
= & (P^{+c.e} ; (\neg\ ok \wedge RT))^f_f \vee (P^{+c.e} ; (ok \wedge \mathit{force}(c.e, Q)))^f_f & \text{[Lemma 25]} \\
= & ((P^f_f \wedge \mathit{possible}(\mathit{ref}, \mathit{ref}', c)); \mathbf{R1_{ct}}(\mathbf{true})) \vee (P^{+c.e} ; (ok \wedge \mathit{force}(c.e, Q)))^f_f & \text{[Def-15,12]} \\
= & ((P^f_f \wedge \mathit{possible}(\mathit{ref}, \mathit{ref}', c)); \mathbf{R1_{ct}}(\mathbf{true})) \vee & \text{[relational calculus]} \\
& (P^{+c.e} ; (ok \wedge (\mathit{try}(c.e, Q) \lhd \mathit{wait} \rhd \mathit{II}_{ct})))^f_f \\
= & ((P^f_f \wedge \mathit{possible}(\mathit{ref}, \mathit{ref}', c)); \mathbf{R1_{ct}}(\mathbf{true})) \vee (P^{+c.e} ; (ok \wedge \neg\ \mathit{wait} \wedge \mathit{II}_{ct}))^f_f \\
& \vee (P^{+c.e} ; (ok \wedge \mathit{try}(c.e, Q) \wedge \mathit{wait}))^f_f & \text{[Lemma 26]} \\
= & \left(\begin{array}{l} ((P^f_f \wedge \mathit{possible}(\mathit{ref}, \mathit{ref}', c)); \mathbf{R1_{ct}}(\mathbf{true})) \vee \\ (P^{+c.e} ; (ok \wedge \neg\ \mathit{wait} \wedge \mathit{II}_{ct}))^f_f \vee \\ ((P^t \wedge \mathit{possible}(\mathit{ref}, \mathit{ref}', c)); (\mathit{wait} \wedge \mathit{wait_com}(c)); (\mathit{wait} \wedge \mathit{II}_{ct}))^f_f \vee \\ ((P^t \wedge \mathit{possible}(\mathit{ref}, \mathit{ref}', c)); (\mathit{wait} \wedge \mathit{term_now_com}(c)); (\neg\ \mathit{wait} \wedge Q_f))^f_f \end{array} \right) & \text{[substitution]} \\
= & \left(\begin{array}{l} ((P^f_f \wedge \mathit{possible}(\mathit{ref}, \mathit{ref}', c)); \mathbf{R1_{ct}}(\mathbf{true})) \vee \mathbf{false} \vee \\ ((P^t \wedge \mathit{possible}(\mathit{ref}, \mathit{ref}', c)); (\mathit{wait} \wedge \mathit{wait_com}(c)); (\mathit{wait} \wedge \mathit{II}_{ct}))^f_f \vee \\ ((P^t \wedge \mathit{possible}(\mathit{ref}, \mathit{ref}', c)); (\mathit{wait} \wedge \mathit{term_now_com}(c)); (\neg\ \mathit{wait} \wedge Q_f))^f_f \end{array} \right) & \text{[substitution]} \\
= & \left(\begin{array}{l} ((P^f_f \wedge \mathit{possible}(\mathit{ref}, \mathit{ref}', c)); \mathbf{R1_{ct}}(\mathbf{true})) \vee \mathbf{false} \vee \\ ((P^t_f \wedge \mathit{possible}(\mathit{ref}, \mathit{ref}', c)); (\mathit{wait} \wedge \mathit{wait_com}(c)); (\mathit{wait} \wedge \neg\ ok \wedge RT)) \vee \\ ((P^t_f \wedge \mathit{possible}(\mathit{ref}, \mathit{ref}', c)); (\mathit{wait} \wedge \mathit{term_now_com}(c)); (\neg\ \mathit{wait} \wedge Q^f_f)) \end{array} \right) & \text{[relational calculus]} \\
= & \left(\begin{array}{l} ((P^f_f \wedge \mathit{possible}(\mathit{ref}, \mathit{ref}', c)); \mathbf{R1_{ct}}(\mathbf{true})) \vee \\ ((P^t_f \wedge \mathit{possible}(\mathit{ref}, \mathit{ref}', c)); (\mathit{wait} \wedge \mathit{wait_com}(c)); \mathbf{R1_{ct}}(\mathbf{true})) \vee \\ ((P^t_f \wedge \mathit{possible}(\mathit{ref}, \mathit{ref}', c)); (\mathit{wait} \wedge \mathit{term_now_com}(c)); (\neg\ \mathit{wait} \wedge Q^f_f)) \end{array} \right) & \text{[relational calculus]} \\
= & \left(\begin{array}{l} ((P^f_f \wedge \mathit{possible}(\mathit{ref}, \mathit{ref}', c)); \mathbf{R1_{ct}}(\mathbf{true})) \vee \\ ((P^t_f \wedge \mathit{possible}(\mathit{ref}, \mathit{ref}', c)); (\mathit{wait} \wedge \mathit{term_now_com}(c)); (\neg\ \mathit{wait} \wedge Q^f_f)) \end{array} \right) &
\end{array}$$

□

Lemma 28. $(P \triangle_c Q)^t_f =$

$$\left(\begin{array}{l} ((P^f_f \wedge \mathit{possible}(\mathit{ref}, \mathit{ref}', c)); \mathbf{R1_{ct}}(\boldsymbol{true})) \vee \\ (P^t_f \wedge (\mathit{possible}(\mathit{ref}, \mathit{ref}', c); \mathit{front}(\mathit{ref}') = \mathit{front}(\mathit{ref})) \wedge \neg\ \mathit{wait}') \vee \\ (P^t_f \wedge \mathit{possible}(\mathit{ref}, \mathit{ref}', c) \wedge \mathit{wait}') \vee \\ ((P^t_f \wedge \mathit{possible}(\mathit{ref}, \mathit{ref}', c)); (\mathit{wait} \wedge \mathit{term\_now\_com}(c)); (\neg\ \mathit{wait} \wedge Q^t_f)) \end{array} \right)$$

Proof.

$$
\begin{array}{ll}
& (P \triangle_c\ Q)^t_f & \text{[Def-17]}\\
= & (\mathbf{R3_{ct}} \circ \mathbf{CSP2_{ct}}(P^{+c}\,;\, interrupt(c, Q)))^t_f & [\mathbf{R3_{ct}} \text{ and substitution}]\\
= & (\mathbf{CSP2_{ct}}(P^{+c}\,;\, interrupt(c, Q)))^t_f & [\mathbf{CSP2_{ct}}]\\
= & ((P^{+c}\,;\, interrupt(c, Q))\,;\, J)^t_f & \text{[J-split]}\\
= & ((P^{+c}\,;\, interrupt(c, Q))^f \vee (ok' \wedge (P^{+c}\,;\, interrupt(c, Q))^t))^t_f & \text{[subs.]}\\
= & (P^{+c}\,;\, interrupt(c, Q))^t_f & \text{[Def-16]}\\
= & (P^{+c}\,;\, \mathbf{CSP1_{ct}}(force(c, Q)))^t_f & [\mathbf{CSP1_{ct}}]\\
= & (P^{+c}\,;\, ((\neg\ ok \wedge RT) \vee (ok \wedge force(c, Q))))^t_f & \text{[relational calculus]}\\
= & (P^{+c}\,;\, (\neg\ ok \wedge RT))^t_f \vee (P^{+c}\,;\, (ok \wedge force(c, Q)))^t_f & \text{[Lemma 25]}\\
= & ((P^f_f \wedge possible(ref,ref',c));\, \mathbf{R1_{ct}}(\mathbf{true})) \vee (P^{+c}\,;\, (ok \wedge force(c,Q)))^t_f & \text{[Def-12,15]}\\
= & \left(\begin{array}{l} ((P^f_f \wedge possible(ref, ref', c));\, \mathbf{R1_{ct}}(\mathbf{true})) \vee \\ (P^{+c}\,;\, (ok \wedge (try(c, Q) \lhd wait \rhd \mathbb{I}_{ct})))^t_f \end{array}\right) & \text{[relational calculus]}\\
= & ((P^f_f \wedge possible(ref, ref', c));\, \mathbf{R1_{ct}}(\mathbf{true})) \vee (P^{+c}\,;\, (ok \wedge \neg\ wait \wedge \mathbb{I}_{ct}))^t_f & \\
& \vee (P^{+c}\,;\, (ok \wedge try(c, Q) \wedge wait))^t_f & [\mathbb{I}_{ct} \text{ and propositional calculus}]\\
= & ((P^f_f \wedge possible(ref, ref', c));\, \mathbf{R1_{ct}}(\mathbf{true})) \vee (P^{+c}\,;\, (ok \wedge \neg\ wait \wedge \mathbb{I}))^t_f & \\
& \vee (P^{+c}\,;\, (ok \wedge try(c, Q) \wedge wait))^t_f & \text{[Lemma 26]}\\
= & \left(\begin{array}{l} ((P^f_f \wedge possible(ref, ref', c));\, \mathbf{R1_{ct}}(\mathbf{true})) \vee (P^{+c}\,;\, (ok \wedge \neg\ wait \wedge \mathbb{I}))^t_f \vee \\ (P \wedge possible(ref, ref', c) \wedge ok' \wedge wait')^t_f \vee \\ ((P \wedge possible(ref, ref', c));(ok \wedge wait \wedge term\_now\_com(c));(\neg\ wait \wedge Q_f))^t_f \end{array}\right) & \\
& & \text{[Def-13 and relational calculus]}\\
= & \left(\begin{array}{l} ((P^f_f \wedge possible(ref, ref', c));\, \mathbf{R1_{ct}}(\mathbf{true})) \vee \\ ((P \wedge (possible(ref, ref', c);\, front(ref') = front(ref)));(ok \wedge \neg\, wait \wedge \mathbb{I}))^t_f \vee \\ (P \wedge possible(ref, ref', c) \wedge ok' \wedge wait')^t_f \vee \\ ((P \wedge possible(ref, ref', c));(ok \wedge wait \wedge term\_now\_com(c));(\neg\ wait \wedge Q_f))^t_f \end{array}\right) & \text{[relational calculus]}\\
= & \left(\begin{array}{l} ((P^f_f \wedge possible(ref,ref',c));\, \mathbf{R1_{ct}}(\mathbf{true})) \vee \\ (P \wedge (possible(ref, ref', c);\, front(ref') = front(ref)) \wedge \neg\ wait')^t_f \vee \\ (P \wedge possible(ref, ref', c) \wedge ok' \wedge wait')^t_f \vee \\ ((P \wedge possible(ref, ref', c));(ok \wedge wait \wedge term\_now\_com(c));(\neg\ wait \wedge Q_f))^t_f \end{array}\right) & \\
& & \text{[substitution and relational calculus]}\\
= & \left(\begin{array}{l} ((P^f_f \wedge possible(ref, ref', c));\, \mathbf{R1_{ct}}(\mathbf{true})) \vee \\ (P^t_f \wedge (possible(ref, ref', c);\, front(ref') = front(ref)) \wedge \neg\ wait') \vee \\ (P^t_f \wedge possible(ref, ref', c) \wedge wait') \vee \\ ((P^t_f \wedge possible(ref, ref', c));(wait \wedge term\_now\_com(c));(\neg\ wait \wedge Q^t_f)) \end{array}\right) &
\end{array}
$$

□

Thus, we finally get the following reactive design for the catastrophic interrupt, simply by applying Lemma 27 and 28 and Theorem 1.

Theorem 24.

$$
(P \triangle_c\ Q) = \mathbf{R_{ct}}\left(\begin{array}{c} \neg\ ((P^f_f \wedge possible(ref, ref', c));\, \mathbf{R1_{ct}}(\mathbf{true})) \wedge \\ \neg\ ((P^t_f \wedge possible(ref,ref',c));(wait \wedge term\_now\_com(c));(\neg\, wait \wedge Q^f_f)) \\ \vdash \\ (P^t_f \wedge (possible(ref, ref', c);\, front(ref') = front(ref)) \wedge \neg\ wait') \vee \\ (P^t_f \wedge possible(ref, ref', c) \wedge wait') \vee \\ ((P^t_f \wedge possible(ref, ref', c));(wait \wedge term\_now\_com(c));(\neg\ wait \wedge Q^t_f)) \end{array}\right)
$$

Proof.

$$
\begin{array}{ll}
& (P \triangle_c\ Q) & \text{[Theorem 1]}\\
=& \mathbf{R_{ct}}(\neg\ (P \triangle_c\ Q)^f_f \vdash (P \triangle_c\ Q)^t_f) & \text{[design]}\\
=& \mathbf{R_{ct}}(\neg\ ok \vee (P \triangle_c\ Q)^f_f \vee (ok' \wedge (P \triangle_c\ Q)^t_f)) & \text{[Lemma 27, 28]}\\
=& \mathbf{R_{ct}}\left(\begin{array}{l}
\neg\ ok \vee \left(\begin{array}{l}((P^f_f \wedge possible(ref, ref', c)); \mathbf{R1_{ct}}(\mathbf{true})) \vee\\ ((P^t_f \wedge possible(ref, ref', c)); (wait \wedge term\_now\_com(c)); (\neg\ wait \wedge Q^f_f))\end{array}\right) \vee\\
\left(ok' \wedge \left(\begin{array}{l}((P^f_f \wedge possible(ref, ref', c)); \mathbf{R1_{ct}}(\mathbf{true})) \vee\\ (P^t_f \wedge (possible(ref, ref', c); front(ref') = front(ref)) \wedge \neg\ wait') \vee\\ (P^t_f \wedge possible(ref, ref', c) \wedge wait') \vee\\ ((P^t_f \wedge possible(ref, ref', c)); (wait \wedge term\_now\_com(c)); (\neg\ wait \wedge Q^t_f))\end{array}\right)\right)
\end{array}\right) & \\
& & \text{[propositional calculus and design]}\\
=& \mathbf{R_{ct}}\left(\begin{array}{c}
\neg\ ((P^f_f \wedge possible(ref, ref', c)); \mathbf{R1_{ct}}(\mathbf{true})) \wedge\\
\neg\ ((P^t_f \wedge possible(ref, ref', c)); (wait \wedge term\_now\_com(c)); (\neg\ wait \wedge Q^f_f))\\
\vdash\\
(P^t_f \wedge (possible(ref, ref', c); front(ref') = front(ref)) \wedge \neg\ wait') \vee\\
(P^t_f \wedge possible(ref, ref', c) \wedge wait') \vee\\
((P^t_f \wedge possible(ref, ref', c)); (wait \wedge term\_now\_com(c)); (\neg\ wait \wedge Q^t_f))
\end{array}\right) &
\end{array}
$$

□

The precondition from the above definition states that the interruptible action P diverges only if the interrupt event c has not occurred, or P is interrupted by c, sequentially composed with the divergence of Q. At any waiting state of P, c cannot be refused. In its postcondition, the first clause states that P terminates without an interrupt, the second that P has not terminated before c, and the third that P is interrupted by c and it sequentially behaves like Q. This reactive design is derived from rigorous calculation of Definition 17, which is fully based on the work in [11]. The validation of this definition is also similar to that in [11] by proving that it respects a number of laws.

For example, the existing step law for the catastrophic interrupt given in [9, 17] still holds, and its proof is fully based on the distributive and eliminative laws, and the approach adopted in [11].

Law 70. $(a \to P) \triangle_c\ Q = (a \to (P \triangle_c\ Q)) \Box (c \to Q)$

4.15.2 Generic interrupt

In the generic interrupt, $P \triangle Q$, Q is executed concurrently with P until either P terminates the execution, or Q performs an interrupt event. However, the approach that we used in Section 4.15.1 cannot be applied here because a generic interrupt is actually a paralleled action rather than a specially sequential action. The characteristic of parallelism of the generic interrupt also inspires us to use the idea of parallel composition in the UTP to construct its reactive design.

First of all, we consider the merge function of timed traces and the sequences of refusals of P and Q.

$$ISync(\langle\rangle, S_2, ref_1, ref_2) = (\langle\rangle, ref_1) \tag{4.82}$$

$$ISync(S_1, \langle\rangle, ref_1, ref_2) = (S_1, ref_1) \tag{4.83}$$

$$\begin{array}{l} ISync(\langle t_1\rangle \frown S_1, \langle t_2\rangle \frown S_2, \langle r_1\rangle \frown ref_1, \langle r_2\rangle \frown ref_2) \\ \quad = (\langle t_1\rangle, \langle r_1 \cap r_2\rangle) \odot ISync(S_1, S_2, ref_1, ref_2) \quad \textit{iff}\ \ t_2 = \langle\rangle \end{array} \tag{4.84}$$

$$ISync(\langle t_1\rangle \frown S_1, \langle t_2\rangle \frown S_2, ref_1, ref_2) = (\langle t_2\rangle \frown S_2, ref_2) \quad \textit{iff}\ \ t_2 \neq \langle\rangle \tag{4.85}$$

In *Circus Time*, we split a failure into a trace and a refusal for the convenience of expression or even simpler mechanisation. Unfortunately, here we have to reunite them again as a pair because they are manipulated together. In addition, the new operator to concatenate a sequence of pairs is given as follows.

$$(S_1, ref_1) \odot (S_2, ref_2) = (S_1 \frown S_2, ref_1 \frown ref_2) \tag{4.86}$$

Note that *ISync* does not support commutativity. The rule (4.82) states that P has no further trace to interact with Q. That is, P may terminate or diverge in practice. The rule (4.83) just describes a same

situation for Q. The rule (4.84) presents the behaviour that no interrupt happens within the current time unit. The rule (4.85) underlines that Q interrupts P.

We also consider the values of ok' and $wait'$ that are determined by whether the interrupt has occurred or not. For example, their values are those of Q if interrupted. Otherwise, we take those of P. Hence, we define two predicates to show whether one trace can interrupt another.

$$enable(tr, 0.tr, 1.tr) \mathrel{\widehat{=}} \exists\, tr_0 \bullet \begin{pmatrix} tr_0 \le diff(1.tr, tr) \wedge \frown\!/front(tr_0) = \langle\rangle \wedge \\ last(tr_0) \neq \langle\rangle \wedge \#tr_0 \le \#diff(0.tr, tr) \end{pmatrix}$$

$$disable(tr, 0.tr, 1.tr) \mathrel{\widehat{=}} \begin{pmatrix} (\frown\!/1.tr = \frown\!/tr) \vee \exists\, tr_0 \bullet \begin{pmatrix} tr_0 \le diff(1.tr, tr) \wedge \\ \frown\!/tr_0 = \langle\rangle \wedge \#diff(0.tr, tr) \le tr_0 \end{pmatrix} \end{pmatrix}$$

The predicate *enable* states that if there exists a subsequence of $diff(1.tr, tr)$, which has not executed any event ($\frown\!/front(tr_0) = \langle\rangle$) except for the last element ($last(tr_0) \neq \langle\rangle$), and meanwhile $0.tr$ has not terminated ($\#tr_0 \le \#diff(0.tr, tr)$), we conclude that $1.tr$ can interrupt $0.tr$ and the merge of them must finish with the rule (4.85). The predicate *disable* states that $1.tr$ cannot interrupt $0.tr$ if, either that $1.tr$ has not executed any external events since tr, or that there exists a subsequence of $diff(1.tr, tr)$, which contains empty traces ($\frown\!/tr = \langle\rangle$) only and whose length is longer or equal to $\#diff(0.tr, tr)$.

We consider the merge predicate of the postcondition first, which describes non-divergent behaviours. If P does not diverge and Q cannot interrupt P, no matter Q can diverge or not, the behaviour will not become divergent. In the meantime the values of ok' and $wait'$ depend on those of P.

$$IM1 \mathrel{\widehat{=}} \begin{pmatrix} IMTR(tr,tr',0.ref,1.ref,ref,ref',0.ref,1.ref) \wedge disable(tr,0.tr,1.tr) \\ \wedge\ ok' = 0.ok \wedge state' = 0.state \wedge wait' = 0.wait \end{pmatrix}$$

$$IMTR(tr, tr', 0.tr, 1.tr, ref, ref', 0.ref, 1.ref) \mathrel{\widehat{=}}$$
$$\left(\begin{pmatrix} diff(tr', tr), \\ ref' - front(ref) \end{pmatrix} = ISync\begin{pmatrix} diff(0.tr, tr), diff(1.tr, tr), \\ 0.ref - front(ref), 1.ref - front(ref) \end{pmatrix}\right)$$

Similarly, if Q does not diverge but does interrupt P, the behaviour is still stable regardless of the state of P.

$$IM2 \mathrel{\widehat{=}} \begin{pmatrix} IMTR(tr,tr',0.ref,1.ref,ref,ref',0.ref,1.ref) \wedge enable(tr,0.tr,1.tr) \\ \wedge\ ok' = 1.ok \wedge state' = 0.state \oplus 1.state \wedge wait' = 1.wait \end{pmatrix}$$

Here, we treat the update of local state very roughly, and it will be further discussed later. For the precondition of the reactive design, we are only interested in the divergence of P if the interrupt has not happened, and the one of Q if it has done. As a result, the divergent behaviour can be captured as follows.

$$\exists \begin{pmatrix} 0.tr, 0.ref, \\ 1.tr, 1.ref \end{pmatrix} \bullet \begin{pmatrix} P_f^f[0.tr, 0.ref/tr', ref'] \wedge Q_f[1.tr, 1.ref/tr', ref'] \\ \wedge\ disable(tr, 0.tr, 1.tr) \wedge \\ IMTR(tr, tr', 0.tr, 1.tr, ref, ref', 0.ref, 1.ref) \end{pmatrix}; \mathbf{R1_{ct}}(\mathbf{true})$$

$$\exists \begin{pmatrix} 0.tr, 0.ref, \\ 1.tr, 1.ref \end{pmatrix} \bullet \begin{pmatrix} P_f[0.tr, 0.ref/tr', ref'] \wedge Q_f^f[1.tr, 1.ref/tr', ref'] \\ \wedge\ enable(tr, 0.tr, 1.tr) \wedge \\ IMTR(tr, tr', 0.tr, 1.tr, ref, ref', 0.ref, 1.ref) \end{pmatrix}; \mathbf{R1_{ct}}(\mathbf{true})$$

Thus, the integrated definition of the interrupt operator is a combination of the above cases, including an extra predicate to tackle the immediate divergence of P or Q. That is, the divergent cases are given in the precondition, and the other are given in the postcondition.

Definition 18.

$$
\begin{array}{l}
P \mathbin{\triangle} Q \mathrel{\widehat{=}} \\
\left(\begin{array}{c}
\neg\left(\left(\left(P_f^f \vee Q_f^f\right) \wedge tr' = tr\right); \mathbf{R1_{ct}}(\mathbf{true})\right) \wedge \\
\neg\, \exists \left(\begin{array}{c} 0.tr, 0.ref, \\ 1.tr, 1.ref \end{array}\right) \bullet \left(\begin{array}{c} P_f^f[0.tr, 0.ref / tr', ref'] \wedge \\ Q_f^t[1.tr, 1.ref / tr', ref'] \wedge \\ disable(tr, 0.tr, 1.tr) \wedge \\ IMTR\left(\begin{array}{c} tr, tr', 0.tr, 1.tr, \\ ref, ref', 0.ref, 1.ref \end{array}\right) \end{array}\right); \mathbf{R1_{ct}}(\mathbf{true}) \wedge \\
\neg\, \exists \left(\begin{array}{c} 0.tr, 0.ref, \\ 1.tr, 1.ref \end{array}\right) \bullet \left(\begin{array}{c} P_f^t[0.tr, 0.ref / tr', ref'] \wedge \\ Q_f^t[1.tr, 1.ref / tr', ref'] \wedge \\ enable(tr, 0.tr, 1.tr) \wedge \\ IMTR\left(\begin{array}{c} tr, tr', 0.tr, 1.tr, \\ ref, ref', 0.ref, 1.ref \end{array}\right) \end{array}\right); \mathbf{R1_{ct}}(\mathbf{true}) \\
\vdash \\
\left(\left(P_f^t ;\, U0(out\alpha P)\right) \wedge \left(Q_f^t ;\, U1(out\alpha Q)\right)\right)_{+\{tr, ref\}} ;\, (IM1 \vee IM2)
\end{array}\right)
\end{array}
$$

Here, we use the $\mathbf{CSP2_{ct}}$**-converge** law [3], $P^t = P^t \vee P^f$ if P is $\mathbf{CSP2_{ct}}$ healthy, to replace P_f and Q_f with P_f^t and Q_f^t respectively.

Through the parallel-by-merge approach, we give the reactive designs of the generic interrupt operator. However, we need to carefully consider the treatment of local variables during any merge operation. In parallel composition, we update the values of the local variables by their names, and usually do not allow any action to update the shared local variables because the result of the update cannot be guaranteed. Here, even without involving shared variables, updating local state is still unpredictable in terms of this reactive design. For example, if an assignment is executed in the interruptible action like the following

$$(Wait\ 2 ;\, x := 1) \mathbin{\triangle} a \rightarrow Skip$$

we do not exactly know whether the assignment happens before or after the interrupt, and whether x should be updated at the merge operation. From Definition 18, if a cannot interrupt the former action, the merge predicate $IM1$ directly uses $state' = 0.state$ to update the local state; otherwise, $IM2$ is employed to simply execute $state' = 0.state \oplus 1.state$ no matter that the occurrence of a is before or after $x := 1$.

A ban on using assignments in the interruptible action can easily solve this problem but make *Circus Time* insufficient to model real-time systems. One possible solution is to use a time stamp to record the time when a variable is updated. In other words, we consider a variable as a tuple consisting of an expression (or name), a value and a time stamp. A detailed scenario is that both actions, e.g., P and Q, obtain a same copy of local variables from their predecessor with initialising all time stamps as zero; a variable is always marked with a relative time once its value has been changed in P; and all variables in Q are marked with the relative time when the first event (or the interrupt) occurs; and thus in the merge operation, the variables from P whose time stamps are greater than that in Q are not considered.

Apart from the ambiguous update of local state, the generic interrupt is unexpectedly different from the catastrophic interrupt even if we make the interrupting action as $c \rightarrow Q$. In fact, their relation can be expressed as

$$P \mathbin{\triangle} (c \rightarrow Q) \sqsubseteq P \mathbin{\triangle_c} Q \tag{4.87}$$

since $P \mathbin{\triangle} (c \rightarrow Q)$ contains more behaviours. For example, with two assumptions on the generic interrupt operator, we could prove Law 71, $(P \mathbin{\triangle} c \rightarrow Q) = P \mathbin{\triangle_c} Q$. Because of these two assumptions, we can easily prove the above refinement relation.

The idea adopted in this section to calculate the definition of catastrophe is to consider Q sequentially composed with P but lifted forward to happen whenever P is waiting for the interaction. However, in *Circus Time* P may execute an event immediately only so that it cannot be interrupted by means of the definition in Theorem 24. For example, the interruptible action in Lemma 29 is not interruptible.

Lemma 29. $((a \rightarrow Skip) \mathbin{\Box} Miracle) \mathbin{\triangle_c} Q = ((a \rightarrow Skip) \mathbin{\Box} Miracle)$

Proof.

$$
\begin{array}{ll}
& ((a \to Skip) \,\Box\, Miracle) \triangle_c\ Q & \text{[Theorem 24]} \\
= \mathbf{R_{ct}} & \left(\begin{array}{c} \neg\ ((((a \to Skip) \,\Box\, Miracle)^f_f \wedge possible(ref, ref', c)); \mathbf{R1_{ct}}(\mathbf{true})) \wedge \\ \neg\ ((((a \to Skip) \,\Box\, Miracle)^t_f \wedge possible(ref, ref', c)); (wait \wedge term\_now\_com(c)); (\neg\, wait \wedge Q^f_f)) \\ \vdash \\ (((a \to Skip) \,\Box\, Miracle)^t_f \wedge (possible(ref, ref', c); front(ref') = front(ref)) \wedge \neg\ wait') \vee \\ (((a \to Skip) \,\Box\, Miracle)^t_f \wedge possible(ref, ref', c) \wedge wait') \vee \\ ((((a \to Skip) \,\Box\, Miracle)^t_f \wedge possible(ref, ref', c)); (wait \wedge term\_now\_com(c)); (\neg\ wait \wedge Q^t_f) \end{array} \right) \\
& & \text{[Theorem 26 and its precondition]} \\
= \mathbf{R_{ct}} & \left(\begin{array}{c} \neg\ ((\mathbf{false} \wedge possible(ref, ref', c)); \mathbf{R1_{ct}}(\mathbf{true})) \wedge \\ \neg\ ((((a \to Skip) \,\Box\, Miracle)^t_f \wedge possible(ref, ref', c)); (wait \wedge term\_now\_com(c)); (\neg\, wait \wedge Q^f_f)) \\ \vdash \\ (((a \to Skip) \,\Box\, Miracle)^t_f \wedge (possible(ref, ref', c); front(ref') = front(ref)) \wedge \neg\ wait') \vee \\ (((a \to Skip) \,\Box\, Miracle)^t_f \wedge possible(ref, ref', c) \wedge wait') \vee \\ ((((a \to Skip) \,\Box\, Miracle)^t_f \wedge possible(ref, ref', c)); (wait \wedge term\_now\_com(c)); (\neg\ wait \wedge Q^t_f) \end{array} \right) \\
& & \text{[Theorem 26 and its precondition]} \\
= \mathbf{R_{ct}} & \left(\begin{array}{c} \mathbf{true} \wedge \\ \neg\ ((((a \to Skip) \,\Box\, Miracle)^t_f \wedge possible(ref, ref', c)); (wait \wedge term\_now\_com(c)); (\neg\, wait \wedge Q^f_f)) \\ \vdash \\ (((a \to Skip) \,\Box\, Miracle)^t_f \wedge (possible(ref, ref', c); front(ref') = front(ref)) \wedge \neg\ wait') \vee \\ (((a \to Skip) \,\Box\, Miracle)^t_f \wedge possible(ref, ref', c) \wedge wait') \vee \\ ((((a \to Skip) \,\Box\, Miracle)^t_f \wedge possible(ref, ref', c)); (wait \wedge term\_now\_com(c)); (\neg\ wait \wedge Q^t_f) \end{array} \right) \\
& & \text{[Theorem 26 and its postcondition]} \\
= \mathbf{R_{ct}} & \left(\begin{array}{c} \neg\ \left(\left(\begin{array}{c} diff(tr', tr) = \langle\langle a \rangle\rangle \wedge state' = state \wedge \\ \neg\ wait' \wedge possible(ref, ref', c) \end{array} \right); (wait \wedge term\_now\_com(c)); (\neg\, wait \wedge Q^f_f) \right) \\ \vdash \\ \left(\begin{array}{c} \neg\ wait' \wedge diff(tr', tr) = \langle\langle a \rangle\rangle \wedge state' = state \wedge \\ (possible(ref, ref', c); front(ref') = front(ref)) \end{array} \right) \vee \\ ((\neg\ wait' \wedge diff(tr', tr) = \langle\langle a \rangle\rangle \wedge state' = state) \wedge possible(ref, ref', c) \wedge wait') \vee \\ \left(\left(\begin{array}{c} \neg\ wait' \wedge diff(tr', tr) = \langle\langle a \rangle\rangle \wedge \\ state' = state \wedge possible(ref, ref', c) \end{array} \right); (wait \wedge term\_now\_com(c)); (\neg\ wait \wedge Q^t_f) \right) \end{array} \right) \\
& & \text{[relational calculus]} \\
= \mathbf{R_{ct}} & \left(\begin{array}{c} \mathbf{true} \\ \vdash \\ \left(\begin{array}{c} \neg\ wait' \wedge diff(tr', tr) = \langle\langle a \rangle\rangle \wedge state' = state \wedge \\ (possible(ref, ref', c); front(ref') = front(ref)) \end{array} \right) \vee \\ \mathbf{false} \vee \mathbf{false} \end{array} \right) & [\mathbf{R1_{ct}}] \\
= \mathbf{R_{ct}}(\mathbf{true} \vdash \neg\ wait' \wedge diff(tr', tr) = \langle\langle a \rangle\rangle \wedge state' = state) & \text{[Theorem 26]} \\
= (a \to Skip) \,\Box\, Miracle &
\end{array}
$$

□

Here, Miracle can force the event a to occur immediately and then terminate. More discussion about the interaction between *Miracle* with other operators can be found in [23]. The behaviour in Lemma 29 can be captured by the first clause in the postcondition in Theorem 24. However, Lemma 29 does not hold if using the generic interrupt, because the event c is able to interrupt as long as it occurs immediately too, via the rule 4.85 in *ISync*.

As a result, the relation between the catastrophic and generic interrupts is complement rather than inclusive. The catastrophic interrupt can deal with the update of local state well but restricts the flexibility of interrupting actions, whereas the generic interrupt can capture the behaviour of actions comprehensively but lacks a simple treatment for local variables in interruptible actions.

To validate the reactive design semantics of a generic interrupt and also underpin the consistency between the catastrophic and generic interrupts, we are about to prove the step law (Law 70) by means of Definition 18

but with necessary constraints. We impose two assumptions on the interruptible action P, which are formally expressed as $P \wedge state' = state$ and $P(tr') \wedge tr_0 \leq tr' \Rightarrow P(tr_0)$ respectively. The former states that P does not change its local state, and the latter states that traces are prefix closed. Based on these two assumptions on the interruptible action, we simply prove

Law 71. $(P \triangle c \rightarrow Q) = P \triangle_c Q$

where the catastrophic interrupt has been proved to preserve Law 70 in Section 4.15.1 or the work [11].

Firstly, we give the reactive design of prefix in Definition 19.

Definition 19. *(Prefix)*

$$c \rightarrow Q = \mathbf{R_{ct}}\left(\begin{array}{c} \neg \left(\left(\begin{array}{c} term\_now\_com(c) \vee \\ terminating\_com(c) \end{array}\right) ; \mathbf{R1_{ct}}(\neg\, wait \wedge Q_f^f)\right) \vdash \\ \left(\begin{array}{c} wait\_com(c) \vee term\_now\_com(c) \\ \vee terminating\_com(c) \end{array}\right) ; \mathbf{R1_{ct}}(\mathbb{I} \lhd wait \rhd Q_f^t) \end{array}\right)$$

$$wait\_com(c) \mathrel{\widehat{=}} wait' \wedge possible(ref, ref', c) \wedge {}^\frown/tr' = {}^\frown/tr \wedge state' = state$$

$$term\_next\_com(c) \mathrel{\widehat{=}} (\neg\, wait' \wedge tr' - tr = \langle\langle c \rangle\rangle \wedge front(ref') = ref \wedge state' = state)$$

Secondly, Law 71 relies on the following lemmas.

Lemma 30. $(((term\_now\_com(c) \vee terminating\_com(c)); P) \wedge tr' = tr) = \mathbf{false}$

This lemma, in fact, is used to remove the immediate divergence from the reactive design in Definition 18, since $c \rightarrow Q$ does not so. The proof of this lemma is simple because c is about to occur in $term\_now\_com(c)$ and $terminating\_com(c)$ that contradict $tr' = tr$.

Lemma 31.

$$\begin{array}{l} \exists\, 0.tr, 0.ref, 1.tr, 1.ref \bullet \\ \left(\begin{array}{c} P_f^f[0.tr, 0.ref/tr', ref'] \wedge (c \rightarrow Q)_f^t[1.tr, 1.ref/tr', ref'] \\ \wedge\ disable(tr, 0.tr, 1.tr) \wedge \\ IMTR(tr, tr', 0.tr, 1.tr, ref, ref', 0.ref, 1.ref) \end{array}\right); \mathbf{R1_{ct}}(\mathbf{true}) \\ = (P_f^f \wedge possible(ref, ref', c)); \mathbf{R1_{ct}}(\mathbf{true}) \end{array}$$

If $(c \rightarrow Q)_f^t$ cannot interrupt P_f^f, it means that the trace truncated from $diff(1.tr, tr)$ executes no external event when participating the merge ($IMTR$), while c is not in any refusal between ref' and ref, or $possible(ref, 1.ref, c)$. Therefore, with regard of the rules 4.83 and 4.84 in *ISync*, we obtain the right part of Lemma 31 with ease.

Lemma 32.

$$\begin{array}{l} \exists\, 0.tr, 0.ref, 1.tr, 1.ref \bullet \\ \left(\begin{array}{c} P_f^t[0.tr, 0.ref/tr', ref'] \wedge (c \rightarrow Q)_f^f[1.tr, 1.ref/tr', ref'] \\ \wedge\ enable(tr, 0.tr, 1.tr) \wedge \\ IMTR(tr, tr', 0.tr, 1.tr, ref, ref', 0.ref, 1.ref) \end{array}\right); \mathbf{R1_{ct}}(\mathbf{true}) \\ = (P_f^t \wedge possible(ref, ref', c)); (wait \wedge term\_now\_com(c)); (\neg\, wait \wedge Q_f^f) \end{array}$$

If $(c \rightarrow Q)_f^f$ can interrupt P_f^t before a divergence, $diff(tr', tr)$ consists of the trace truncated from $diff(0.tr', tr)$ and a sequentially immediate execution of $c \rightarrow Q$. The proof of Lemma 32 is also straightforward with our two assumptions.

In combination with Lemma 30, 31 and 32, we can prove the equivalence of the preconditions between $P \triangle c \rightarrow Q$ and $P \triangle_c Q$. Proving the equality of the postcondition between the two operators is very similar to Lemma 31 and 32. That is, the predicate with $IM1$ in Definition 18 corresponds to the first two predicates of the postcondition in Theorem 24, the predicate with $IM2$ directly corresponds to the third predicate in the postcondition of catastrophe. Unlike the alphabet extension in Theorem 24, the merge of refusals in Definition 18 does not need to consider whether P terminates or not. In brief, the generic interrupt holds the step law, but subject to two restrictions, when taking advantage of Law 71. As a matter of fact, the

generic interrupt can hold the step law only if P does not change its local state. Another assumption is solely for Law 71.

The generic interrupt in *Circus Time* can hold other existing laws in CSP. For example, since *Stop* offers no external event, it can never interrupt any action. Similarly, if *Stop* is interruptible, only interrupt can occur.

Law 72. $(P \triangle Stop) = P = (Stop \triangle P)$

If *Skip* is the interrupting action, similar to *Stop*, the interrupt always behaves just like the interruptible action.

Law 73. $P \triangle Skip = P$

However, in *Circus Time*, *Skip* can be interrupted because we can allow events to happen without any delay, which can even occur prior to the start of *Skip*.

Law 74. $Skip \triangle P \sqsubseteq Skip$

In addition, the divergent action cannot be cured by interrupting it, or it is not safe to specify a divergent action after the interrupt.

Law 75. $(P \triangle Chaos) = Chaos = (Chaos \triangle P)$

The proofs of Laws 72-75 can be found in Appendix E.

4.15.3 Timed Interrupt

A timed interrupt, $P \triangle_d Q$, allows P to run for no more than a particular length of time, and then performs an interrupt to pass the control of the the process to Q. Compared with the event-driven interrupt where the environment of the process can prevent the interrupt event from happening, this time-driven interrupt cannot be avoided (if P does not terminate before time d) since its environment is not involved. As a matter of fact, the timed interrupt can be defined via the event-driven interrupt and hiding as follow

Definition 20. $P \triangle_d Q \mathrel{\widehat=} (P \triangle Wait\ d ; (e \rightarrow Q)) \setminus \{e\} \qquad e \notin \alpha(P) \cup \alpha(Q)$

where the special event e becomes urgent to interrupt P immediately after d time units.

However, to avoid the complex semantics introduced by hiding, we directly give its definition to describe the behaviour of the timed interrupt, rather than calculating its reactive design from Definition 20.

Definition 21.

$$P \triangle_d Q \mathrel{\widehat=} \left(\begin{array}{c} \neg\, ((P_f^f \wedge \#tr' - \#tr \leq d) ; \mathbf{R1_{ct}}(\mathbf{true})) \wedge \\ \neg\, ((P_f^t \wedge \#tr' - \#tr = d) ; (wait \wedge \mathbb{I}^{\neg wait} \wedge \neg\, wait') ; Q_f^f) \\ \vdash \\ (P_f^t \wedge \#tr' - \#tr \leq d) \vee \\ ((P_f^t \wedge \#tr' - \#tr = d) ; (wait \wedge \mathbb{I}^{\neg wait} \wedge \neg\, wait') ; Q_f^t) \end{array} \right)$$

In fact, the timed interrupt is quite like a kind of sequential composition because the interrupting action has no influence on the interruptible action. The postcondition in the above definition states that, within d time units, we can only observe P, or Q can take over the program control at d. For the divergent cases, we consider that P diverges within d, otherwise Q diverges after d.

Apart from the usual laws for association, and distribution through nondeterministic choice, it also satisfies the distributive law through external choice.

Law 76. $(P \,\Box\, Q) \triangle_d R = (P \triangle_d R) \,\Box\, (Q \triangle_d R)$

Furthermore, any step law only changes the interruptible action and the time d. For example, we can eliminate the timed interrupt by the following laws.

Law 77. $Stop \triangle_d P = Wait\ d ; P$

Law 78. $Skip \triangle_d P = Skip \;\; if \;\; d > 0$

Law 79. $(Wait\ d ; P) \triangle_{d+d'} Q = Wait\ d ; (P \triangle_{d'} Q)$

Law 80. $(Wait\ (d + d') ; P) \triangle_d Q = Wait\ d ; Q$

The proof of the above laws can be found in Appendix E.

5 Application of Reactive Designs

One of major differences between the original and new *Circus Time* theories is the introduction of *Miracle* in the semantic model. The action *Miracle* itself is a very unusual action since it expresses an action that has not started yet. Such an action has not been explored in the original *Circus Time* and does not exist in the standard CSP models. However, *Miracle* is very useful as a mathematical abstraction in reasoning about properties of a system.

Since the reactive design semantics exposes the pre-postcondition semantics, it provides us with a more concise, readable and uniform UTP semantics, and helps us understand the behaviours of some subtle processes. For example, the interaction between *Miralce* and other operators has not been fully explored particularly in a timed environment, even though the work in [24] has given a remarkable insight to some unusual processes which involve *Miracle* in an untimed environment.

5.1 Prefix and *Miracle*

The action *Miracle* has miraculous behaviour that simply denotes an unstarted state. Therefore, it should never appear during the execution of an action. As a result, the newly established reactive design of sequential composition can help us to figure out the exact behaviour of the combination of a simple prefix and *Miracle*.

Theorem 25.

$$c.e \rightarrow Miracle = \mathbf{R_{ct}}(\mathbf{true} \vdash (wait' \land possible(tr, tr', c) \land \frown/tr' = \frown/tr \land state' = state))$$

Proof.

$$
\begin{array}{ll}
& c.e \rightarrow Miracle \qquad [\text{property of Prefix}] \\
= & \mathbf{R_{ct}}\left(\begin{array}{c} \neg \left(\mathbf{R1_{ct}}\left(\begin{array}{l} wait\_com(c) \lor term\_now\_com(c.e) \\ \lor\, terminating\_com(c.e) \end{array}\right); \mathbf{R1_{ct}}(\neg\, wait \land \mathbf{R2_{ct}}(Miracle_f^f))\right) \\ \vdash \\ \mathbf{R1_{ct}}\left(\begin{array}{l} wait\_com(c) \lor term\_now\_com(c.e) \\ \lor\, terminating\_com(c.e) \end{array}\right); \mathbf{R1_{ct}}(\mathbb{I} \lhd wait \rhd \mathbf{R2_{ct}}(Miracle_f^t)) \end{array}\right) \\
& \qquad [Miracle(\mathbf{??})] \\
= & \mathbf{R_{ct}}\left(\begin{array}{c} \neg \left(\mathbf{R1_{ct}}\left(\begin{array}{l} wait\_com(c) \lor term\_now\_com(c.e) \\ \lor\, terminating\_com(c.e) \end{array}\right); \mathbf{R1_{ct}}(\neg\, wait \land \mathbf{R2_{ct}}(\mathbf{false}))\right) \\ \vdash \\ \mathbf{R1_{ct}}\left(\begin{array}{l} wait\_com(c) \lor term\_now\_com(c.e) \\ \lor\, terminating\_com(c.e) \end{array}\right); \mathbf{R1_{ct}}(\mathbb{I} \lhd wait \rhd \mathbf{R2_{ct}}(\mathbf{false})) \end{array}\right) \\
& \qquad [\mathbf{R2_{ct}}(\mathbf{false}) = \mathbf{false} \text{ and relational calculus}] \\
= & \mathbf{R_{ct}}(\mathbf{true} \vdash \mathbf{R1_{ct}}(wait\_com(c) \lor term\_now\_com(c.e) \lor terminating\_com(c.e))); \mathbf{R1_{ct}}(\mathbb{I} \land wait)) \\
& \qquad [\text{relational calculus}] \\
= & \mathbf{R_{ct}}\left(\mathbf{true} \vdash \left(\begin{array}{c} (\mathbf{R1_{ct}}(wait\_com(c)); \mathbf{R1_{ct}}(\mathbb{I} \land wait)) \lor \\ (\mathbf{R1_{ct}}(term\_now\_com(c.e); \mathbf{R1_{ct}}(\mathbb{I} \land wait)) \lor \\ (\mathbf{R1_{ct}}(terminating\_com(c.e)); \mathbf{R1_{ct}}(\mathbb{I} \land wait)) \end{array}\right)\right) \qquad [terminating\_com(4.59)] \\
= & \mathbf{R_{ct}}\left(\mathbf{true} \vdash \left(\begin{array}{c} (\mathbf{R1_{ct}}(wait\_com(c)); \mathbf{R1_{ct}}(\mathbb{I} \land wait)) \lor \\ (\mathbf{R1_{ct}}(term\_now\_com(c.e); \mathbf{R1_{ct}}(\mathbb{I} \land wait)) \lor \\ (\mathbf{R1_{ct}}(wait\_com(c); term\_next\_com(c.e)); \mathbf{R1_{ct}}(\mathbb{I} \land wait)) \end{array}\right)\right) \\
& \qquad [wait' \text{ in } term\_now\_com \text{ and } term\_next\_com \text{ is } \mathbf{false} \text{ and relational calculus}] \\
= & \mathbf{R_{ct}}(\mathbf{true} \vdash (\mathbf{R1_{ct}}(wait\_com(c)); \mathbf{R1_{ct}}(\mathbb{I} \land wait)) \lor \mathbf{false} \lor \mathbf{false}) \qquad [wait\_com(4.55) \text{ and prop. cal.}] \\
= & \mathbf{R_{ct}}\left(\mathbf{true} \vdash \mathbf{R1_{ct}}\left(\begin{array}{l} wait' \land possible(tr, tr', c) \land \\ \frown/tr' = \frown/tr \land state' = state \end{array}\right); \mathbf{R1_{ct}}(\mathbb{I} \land wait)\right) \quad [\mathbf{R1_{ct}}, \text{ rel. cal. and unit law}] \\
= & \mathbf{R_{ct}}(\mathbf{true} \vdash \mathbf{R1_{ct}}(wait' \land possible(tr, tr', c) \land \frown/tr' = \frown/tr \land state' = state)) \qquad [\text{Law 2}] \\
= & \mathbf{R_{ct}}(\mathbf{true} \vdash wait' \land \frown/tr' = \frown/tr \land possible(tr, tr', c) \land state' = state)
\end{array}
$$

□

This theorem states that, if this action starts, it will wait for interaction with its environment ($wait' = \mathbf{true}$), but will never actually perform any event ($\frown/tr' = \frown/tr$) even if the event $c.e$ has been offered. This action is different from that of the standard CSP failures-divergences model in which one of the assumptions requires that, if an event is not in the refusal set, the process is always willing to execute the event. This theorem also shows that, compared with the failures-divergences model of CSP, the semantics of *Circus Time* is rather loose.

5.2 External Choice and Miracle

Another unusual behaviour is that of the external choice of *Miracle* with a simple prefix. Below, we give two auxiliary laws about the components in the definitions of prefix and external choice. And the proof of the two lemmas is straightforward and can be found in [23]..

Lemma 33. $term\_now \land wait\_com(c) = \mathbf{false}$

Lemma 34. $term\_now \land terminating\_com(c) = \mathbf{false}$

Theorem 26.

$$(c.e \rightarrow Skip) \Box Miracle = \mathbf{R_{ct}}(\mathbf{true} \vdash (diff(tr', tr) = \langle\langle c.e\rangle\rangle \land state' = state \land \neg\, wait'))$$

Proof.

$$
\begin{array}{ll}
& (c.e \rightarrow Skip) \Box Miracle \\
& \qquad [\text{Simple prefix(Theorem 11) and } Miracle] \\
= & \mathbf{R_{ct}}(\mathbf{true} \vdash (wait\_com(c) \lor term\_now\_com(c.e) \lor terminating\_com(c.e))) \Box \mathbf{R_{ct}}(\mathbf{true} \vdash \mathbf{false}) \\
& \qquad [\Box(\text{Theorem 15}) \text{ and relational calculus}] \\
= & \mathbf{R_{ct}}\left(\begin{array}{c} \mathbf{true} \vdash \\ \left((wait\_com(c) \lor term\_now\_com(c.e) \lor terminating\_com(c.e)) \land \mathbf{false} \land wait' \land \frown/tr' = \frown/tr\right) \lor \\ (Diff((c.e \rightarrow skip)^t_f, \mathbf{false}) \land ((c.e \rightarrow Skip)^t_f \lor \mathbf{false})) \end{array}\right) \\
& \qquad [\text{propositional calculus}] \\
= & \mathbf{R_{ct}}(\mathbf{true} \vdash Diff((c.e \rightarrow Skip)^t_f, \mathbf{false}) \land (c.e \rightarrow Skip)^t_f) \qquad [Diff\,(4.60)] \\
= & \mathbf{R_{ct}}(\mathbf{true} \vdash (((c.e \rightarrow Skip)^t_f \land \mathbf{false} \land wait' \land \frown/tr' = \frown/tr);\, term\_next \lor term\_now) \land (c.e \rightarrow Skip)^t_f) \\
& \qquad [\text{relational calculus}] \\
= & \mathbf{R_{ct}}(\mathbf{true} \vdash term\_now \land (c.e \rightarrow Skip)^t_f) \qquad [\text{postcondition in prefix}] \\
= & \mathbf{R_{ct}}(\mathbf{true} \vdash term\_now \land (wait\_com(c) \lor term\_now\_com(c.e) \lor terminating\_com(c.e))) \qquad [\text{prop. cal.}] \\
= & \mathbf{R_{ct}}\left(\mathbf{true} \vdash \left(\begin{array}{c} (term\_now \land wait\_com(c)) \lor (term\_now \land term\_now\_com(c.e)) \\ \lor\, (term\_now \land terminating\_com(c.e)) \end{array}\right)\right) \\
& \qquad [\text{Lemma 33 and 34 }] \\
\\
= & \mathbf{R_{ct}}(\mathbf{true} \vdash \mathbf{false} \lor (term\_now \land term\_now\_com(c.e)) \lor \mathbf{false}) \qquad [term\_now(4.61),\ term\_now\_com(4.57)] \\
= & \mathbf{R_{ct}}\left(\mathbf{true} \vdash \left(\begin{array}{c} ((\neg\, wait' \land tr' = tr) \lor (head(diff(tr', tr)) \neq \langle\rangle)) \land \\ (\ \neg\, wait' \land diff(tr', tr) = \langle\langle c.e\rangle\rangle \land front(ref') = front(ref) \land state' = state\ ) \end{array}\right)\right) \\
& \qquad [\text{property of sequences and propositional calculus}] \\
= & \mathbf{R_{ct}}\left(\mathbf{true} \vdash (\ \neg\, wait' \land diff(tr', tr) = \langle\langle c.e\rangle\rangle \land front(ref') = front(ref) \land state' = state\ )\right) \qquad [\mathbf{R1_{ct}}] \\
= & \mathbf{R_{ct}}(\mathbf{true} \vdash (\neg\, wait' \land diff(tr', tr) = \langle\langle c.e\rangle\rangle \land state' = state))
\end{array}
$$

□

This above theorem states that this action performs the event $c.e$ and terminates immediately. There is no state in which the action is waiting for the environment to offer $c.e$. It simply occurs instantly since there is no observation in which $tr' = tr$. This violates another important assumption of the standard CSP models where traces are prefix closed.

5.3 *Skip* in External Choice

From the definition of prefix in Section 4.6, an event may occur instantly if its environment is ready. However, the action in Theorem 26 actually excludes other choices and preserves the instantaneity of the event only. In other words, *Miracle* is able to make external events become *urgent*.

The action *Skip* can also make events become urgent. For example, if putting *Skip* with a simple prefix in an external choice, we can have the following theorem.

Theorem 27.

$$(c.e \to Skip) \,\Box\, Skip = Skip \vee \mathbf{R_{ct}}\left(\mathbf{true} \vdash \left(\ \neg\, wait' \wedge state' = state \wedge diff(tr', tr) = \langle\langle c.e\rangle\rangle\ \right)\right)$$

Proof.

$$\begin{aligned}
&(c.e \to Skip) \,\Box\, Skip && \text{[Simple prefix (Theorem 11) and } Skip(\mathbf{??})\text{]}\\
&= \mathbf{R_{ct}}\left(\mathbf{true} \vdash \begin{pmatrix} wait_com(c) \vee term_now_com(c.e) \\ \vee\ terminating_com(c.e)\end{pmatrix}\right) \Box\, \mathbf{R_{ct}}\left(\mathbf{true} \vdash \begin{pmatrix} \neg\, wait' \wedge tr' = tr \\ \wedge\ state' = state\end{pmatrix}\right) \\
& && \text{[def-}\Box\text{ (Theorem 15) and relational calculus]}\\
&= \mathbf{R_{ct}}\left(\mathbf{true} \vdash \begin{pmatrix} ((c.e \to Skip)^t_f \wedge Skip^t_f \wedge wait' \wedge \frown\!/tr' = \frown\!/tr) \vee \\ (Diff((c.e \to Skip)^t_f, Skip^t_f) \wedge ((c.e \to Skip)^t_f \vee Skip^t_f))\end{pmatrix}\right) && [wait' \text{ is } \mathbf{false} \text{ in } Skip^t_f]\\
&= \mathbf{R_{ct}}\left(\ \mathbf{true} \vdash \mathbf{false} \vee (Diff((c.e \to Skip)^t_f, Skip^t_f) \wedge ((c.e \to Skip)^t_f \vee Skip^t_f))\ \right) && [Diff\,(4.60)]\\
&= \mathbf{R_{ct}}\left(\ \mathbf{true} \vdash \left(\begin{pmatrix} (c.e \to Skip)^t_f \wedge Skip^t_f \\ \wedge\ wait' \wedge \frown\!/tr' = \frown\!/tr\end{pmatrix} ;\ term_next \vee term_now\right) \wedge ((c.e \to Skip)^t_f \vee Skip^t_f)\ \right) \\
& && [wait' \text{ is } \mathbf{false} \text{ in } Skip^t_f]\\
&= \mathbf{R_{ct}}\left(\ \mathbf{true} \vdash (\mathbf{false}\,;\ term_next \vee term_now) \wedge ((c.e \to Skip)^t_f \vee Skip^t_f)\ \right) && \text{[relational calculus]}\\
&= \mathbf{R_{ct}}(\mathbf{true} \vdash term_now \wedge ((c.e \to Skip)^t_f \vee Skip^t_f)) && \text{[propositional calculus]}\\
&= \mathbf{R_{ct}}(\mathbf{true} \vdash (term_now \wedge (c.e \to Skip)^t_f) \vee (term_now \wedge Skip^t_f)) && \text{[Properties of } \mathbf{R_{ct}}\text{]}\\
&= \mathbf{R_{ct}}(\mathbf{true} \vdash term_now \wedge Skip^t_f) \vee \mathbf{R_{ct}}(\mathbf{true} \vdash term_now \wedge (c.e \to Skip)^t_f) && [term_now(4.61) \text{ and subs.}]\\
&= \mathbf{R_{ct}}\left(\mathbf{true} \vdash \begin{pmatrix} ((\neg\, wait' \wedge tr' = tr) \vee head(diff(tr', tr)) \neq \langle\rangle) \\ \wedge\ (\neg\, wait' \wedge tr' = tr \wedge state' = state)\end{pmatrix}\right) \vee \mathbf{R_{ct}}(\mathbf{true} \vdash term_now \wedge (c.e \to Skip)^t_f) \\
& && \text{[Properties of sequences and propositional calculus]}\\
&= \mathbf{R_{ct}}(\mathbf{true} \vdash \neg\, wait' \wedge tr' = tr \wedge state' = state) \vee \mathbf{R_{ct}}(\mathbf{true} \vdash term_now \wedge (c.e \to Skip)^t_f) && [Skip(\mathbf{??})]\\
&= Skip \vee \mathbf{R_{ct}}(\mathbf{true} \vdash term_now \wedge (c.e \to Skip)^t_f) && \text{[postcondition of prefix]}\\
&= Skip \vee \mathbf{R_{ct}}(\mathbf{true} \vdash term_now \wedge (wait_com(c) \vee term_now_com(c.e) \vee terminating_com(c))) \\
& && \text{[Lemma 33,34]}
\end{aligned}$$

$$\begin{aligned}
&= Skip \vee \mathbf{R_{ct}}(\mathbf{true} \vdash \mathbf{false} \vee (term_now \wedge term_now_com(c.e)) \vee \mathbf{false}) \\
& && [term_now(4.61) \text{ and } term_now_com(4.57)]\\
&= Skip \vee \mathbf{R_{ct}}\left(\mathbf{true} \vdash \left(\begin{array}{c} ((\neg\, wait' \wedge tr' = tr) \vee (head(diff(tr', tr)) \neq \langle\rangle)) \wedge \\ (\ \neg\, wait' \wedge diff(tr', tr) = \langle\langle c.e\rangle\rangle \wedge front(ref') = front(ref) \wedge state' = state\ )\end{array}\right)\right) \\
& && \text{[property of sequences and propositional calculus]}\\
&= Skip \vee \mathbf{R_{ct}}(\mathbf{true} \vdash \neg\, wait' \wedge diff(tr', tr) = \langle\langle c.e\rangle\rangle \wedge state' = state)
\end{aligned}$$

□

Even though *Skip* cannot guarantee that $c.e$ must occur immediately, it excludes the possible behaviour that the event is waiting for the interaction from its environment.

5.4 Hiding in Recursion

There is a very subtle law in the CSP theory about hiding and recursion as

$$(\mu\, P \bullet c \to P) \setminus \{c\} = Chaos \tag{5.88}$$

which is difficult to be proved using their original UTP definitions. However, the newly-defined reactive designs allow us to prove this law with little effort. To prove this law, we use the *Kleene* theorem rather than the traditional definition of the weakest fixed point to calculate the recursive design in the recursively reactive design.

Theorem 28. *(Kleene fixed point theorem)*
If F is continuous [2]*, then $\mu\, X \bullet F(X) = \bigsqcup_{n=0}^{\infty} F^n(\mathbf{true})$ where $F^0(X) \mathrel{\widehat{=}} \mathbf{true}$, and $F^{n+1} \mathrel{\widehat{=}} F(F^n(X))$.*

This theorem states a normal form for programs that contain recursion. First of all, the behaviour of a recursive program is expressed as an *infinite* sequence of predicates $\{F^i \mid i \in \mathbb{N}\}$ and each F^i is a finite normal form. Since each F^{i+1} is defined by its previous expression, F^{i+1} is potentially stronger if $F^i \sqsubseteq F^{i+1}$. If i is large enough, the exact behaviour of the program can be captured by the *least upper bound* of the infinite sequence, written $\bigsqcup_{n=0}^{\infty} F^n(\mathbf{true})$.

In addition, we are able to prove a similar theorem for the strongest fixed point of F.

Theorem 29. *If F is continuous, $\nu X \bullet F(X) = \bigsqcap_{n=0}^{\infty} F^n(\mathbf{false})$*

Proof.

$$
\begin{array}{ll}
& \nu X \bullet F(X) \qquad\qquad [\text{def of } \nu] \\
= & \neg\ \mu\, X \bullet \neg\ F(\neg\ X) \qquad\qquad [\text{Theorem 28}] \\
= & \neg \bigsqcup_{n=0}^{\infty} (\lambda\, X \bullet \neg\ F(\neg\ X))^n(\mathbf{true}) \qquad\qquad [\text{relational calculus}] \\
= & \bigsqcap_{n=0}^{\infty} \neg\ (\lambda\, X \bullet \neg\ F(\neg\ X))^n(\mathbf{true}) \qquad\qquad [\text{predicate calculus}] \\
= & \bigsqcap_{n=0}^{\infty} F^n(\mathbf{false})
\end{array}
$$

□

Now, firstly, we calculate the reactive design of a single call of $\mu\, P \bullet c \to P$ as Law 81. This is simply achieved by Theorem **??**.

Law 81.

$$
c \to P = \mathbf{R_{ct}} \begin{pmatrix} \neg\ ((\mathit{term\_now\_com}(c) \vee\ \mathit{terminating\_com}(c)); \mathbf{R1_{ct}}(\neg\ \mathit{wait} \wedge \mathbf{R2_{ct}}(P_f^f))) \\ \vdash \\ \mathit{wait\_com}(c) \vee (\mathit{term\_now\_com}(c)\,;\, \mathbf{R1_{ct}}(\neg\ \mathit{wait} \wedge \mathbf{R2_{ct}}(P_f^t))) \vee \\ (\mathit{terminating\_com}(c)\,;\, \mathbf{R1_{ct}}(\neg\ \mathit{wait} \wedge \mathbf{R2_{ct}}(P_f^t))) \end{pmatrix}
$$

Secondly, via Theorem 18 and Law 81, we let $X = \neg\ P_f^f$ and $Y = P_f^t$, then

$$
F(X, Y) = \neg((\mathit{term\_now\_com}(c) \vee\ \mathit{terminating\_com}(c)); \mathbf{R1_{ct}}(\neg\ \mathit{wait} \wedge \mathbf{R2_{ct}}(\neg\ X))) \tag{5.89}
$$

$$
G(X, Y) = \begin{pmatrix} \mathit{wait\_com}(c) \vee (\mathit{term\_now\_com}(c)\,;\, \mathbf{R1_{ct}}(\neg\ \mathit{wait} \wedge \mathbf{R2_{ct}}(Y))) \vee \\ (\mathit{terminating\_com}(c)\,;\, \mathbf{R1_{ct}}(\neg\ \mathit{wait} \wedge \mathbf{R2_{ct}}(Y))) \end{pmatrix} \tag{5.90}
$$

As a result, the weakest fixed point of $(\mu\, P \bullet c \to P) \setminus \{c\}$ can be calculated by the following law.

Law 82.

$$
\mu(X, Y) \bullet \mathbf{R_{ct}}(F(X, Y) \vdash G(X, Y)) = \mathbf{R_{ct}}((\nu X \bullet F(X, Y)) \vdash Q)
$$

Proof.

$$
\begin{array}{ll}
& \mu(X, Y) \bullet \mathbf{R_{ct}}(F(X, Y) \vdash G(X, Y)) \qquad\qquad [\text{Theorem 18}] \\
= & \mathbf{R_{ct}}(\mu(X, Y) \bullet F(X, Y) \vdash G(X, Y)) \qquad\qquad [\text{Law 68}] \\
= & \mathbf{R_{ct}}((\nu X \bullet F(X, Y)) \vdash Q)
\end{array}
$$

□

[2] A function is *continuous* only if its value at a limit point can be determined from its values on a sequence converging to that point. Also, a continuous function is monotonic.

Note that, since the postcondition of the recursive design has no influence on the final result, we here simply use Q to denote it and never unfold it in the later proof.

Next, we calculate the strongest fixed point of F by means of the *Kleene* theorem. Before starting to prove the law (5.88), we give some useful properties where Lemma 35 and 36 states that they are $\mathbf{R2_{ct}}$ healthy, and Lemma 37 is a sequence property. The proofs of these three lemmas can be found in [23]. Lemma 38 calculates the precondition of $\mu\, P \bullet c \rightarrow P$.

Lemma 35. $\mathbf{R2_{ct}}(\mathit{term\_now\_com}(c)) = \mathit{term\_now\_com}(c)$

Lemma 36. $\mathbf{R2_{ct}}(\mathit{terminating\_com}(c)) = \mathit{terminating\_com}(c)$

Lemma 37.

$$\begin{pmatrix} \mathit{diff}(tr', tr) = \langle\langle c\rangle\rangle \wedge \\ \mathit{front}(ref') = \mathit{front}(ref) \end{pmatrix}; \begin{pmatrix} \mathit{diff}(tr', tr) = \langle\langle c\rangle\rangle \wedge \\ \mathit{front}(ref') = \mathit{front}(ref) \end{pmatrix} = (\mathit{diff}(tr', tr) = \langle\langle c, c\rangle\rangle \wedge \mathit{front}(ref') = \mathit{front}(ref))$$

Lemma 38.

$$\neg\, \nu\, (\lambda\, X \bullet \neg\, (\!(\ \mathit{term\_now\_com}(c) \vee \mathit{terminating\_com}(c)\ )\!)\, ;\, \mathbf{R1_{ct}}(\neg\, \mathit{wait} \wedge \mathbf{R2_{ct}}(\neg\, X)))) =$$
$$\begin{pmatrix} \prod_{n=0}^{\infty} (\!(\ \mathit{front}(ref') = \mathit{front}(ref) \wedge \mathit{diff}(tr', tr) = \langle\langle c\rangle^n\rangle \wedge \mathit{state}' = \mathit{state}\ )\!)\, ;\, \mathbf{R1_{ct}}(\mathbf{true}) \vee \\ \neg \prod_{n=0}^{\infty} \neg\, (\!(\ \mathit{term\_now\_com}(c) \vee \mathit{terminating\_com}(c)\ )\!)\, ;\, \mathbf{R1_{ct}}(\neg\, \mathit{wait} \wedge \mathbf{R2_{ct}}(\neg\, \mathbf{false}))\big)^n \end{pmatrix}$$

Proof.

$$\begin{aligned}
&\neg\, \nu(\lambda\, X \bullet \neg\, ((\mathit{term_now_com}(c) \vee \mathit{terminating_com}(c)); \mathbf{R1_{ct}}(\neg\, \mathit{wait} \wedge \mathbf{R2_{ct}}(\neg\, X)))) && \text{[Theorem 29]}\\
&= \neg \prod_{n=0}^{\infty} (\lambda\, X \bullet \neg((\mathit{term_now_com}(c) \vee \mathit{terminating_com}(c)); \mathbf{R1_{ct}}(\neg\, \mathit{wait} \wedge \mathbf{R2_{ct}}(\neg\, X))))^n(\mathbf{false}) && \text{[subs.]}\\
&= \neg \prod_{n=0}^{\infty} \neg\, (\!(\ \mathit{term_now_com}(c) \vee \mathit{terminating_com}(c)\ )\!)\, ;\, \mathbf{R1_{ct}}(\neg\, \mathit{wait} \wedge \mathbf{R2_{ct}}(\neg\, \mathbf{false}))\big)^n && \text{[unfold } \textstyle\prod\text{]}\\
&= \neg \begin{pmatrix} \neg\, (\!(\ \mathit{term_now_com}(c) \vee \mathit{terminating_com}(c)\ )\!)\, ;\, \mathbf{R1_{ct}}(\neg\, \mathit{wait} \wedge \mathbf{R2_{ct}}(\neg\, \mathbf{false}))\big) \vee \\ \neg\, (\!(\ \mathit{term_now_com}(c) \vee \mathit{terminating_com}(c)\ )\!)\, ;\, \mathbf{R1_{ct}}(\neg\, \mathit{wait} \wedge \mathbf{R2_{ct}}(\neg\, \mathbf{false}))\big)^1 \vee \\ \prod_{n=2}^{\infty} \neg\, (\!(\ \mathit{term_now_com}(c) \vee \mathit{terminating_com}(c)\ )\!)\, ;\, \mathbf{R1_{ct}}(\neg\, \mathit{wait} \wedge \mathbf{R2_{ct}}(\neg\, \mathbf{false}))\big)^n \end{pmatrix} \\
&&& \text{[property of } \textstyle\prod \text{ and Lemma 35,36]}\\
&= \begin{pmatrix} \prod_{n=0}^{\infty} ((\mathit{term_now_com}(c))^n; \mathbf{R1_{ct}}(\neg\, \mathbf{false})) \vee \\ \neg \prod_{n=0}^{\infty} \neg\, (\!(\ \mathit{term_now_com}(c) \vee \mathit{terminating_com}(c)\ )\!)\, ;\, \mathbf{R1_{ct}}(\neg\, \mathit{wait} \wedge \mathbf{R2_{ct}}(\neg\, \mathbf{false}))\big)^n \end{pmatrix} \\
&&& [\mathit{term_now_com}(4.57)]\\
&= \begin{pmatrix} \prod_{n=0}^{\infty} (\!(\!(\ \neg\, \mathit{wait}' \wedge \mathit{diff}(tr', tr) = \langle\langle c\rangle\rangle \wedge \mathit{front}(ref') = \mathit{front}(ref) \wedge \mathit{state}' = \mathit{state}\ )\!)^n\, ;\, \mathbf{R1_{ct}}(\neg\, \mathbf{false})) \\ \vee \neg \prod_{n=0}^{\infty} \neg\, (\!(\ \mathit{term_now_com}(c) \vee \mathit{terminating_com}(c)\ )\!)\, ;\, \mathbf{R1_{ct}}(\neg\, \mathit{wait} \wedge \mathbf{R2_{ct}}(\neg\, \mathbf{false}))\big)^n \end{pmatrix} \\
&&& \text{[Lemma 37 and relational calculus]}\\
&= \begin{pmatrix} \prod_{n=0}^{\infty} (\!(\ \mathit{front}(ref') = \mathit{front}(ref) \wedge \mathit{diff}(tr', tr) = \langle\langle c\rangle^n\rangle \wedge \mathit{state}' = \mathit{state}\ )\!)\, ;\, \mathbf{R1_{ct}}(\neg\, \mathbf{false})) \vee \\ \neg \prod_{n=0}^{\infty} \neg\, (\!(\ \mathit{term_now_com}(c) \vee \mathit{terminating_com}(c)\ )\!)\, ;\, \mathbf{R1_{ct}}(\neg\, \mathit{wait} \wedge \mathbf{R2_{ct}}(\neg\, \mathbf{false}))\big)^n \end{pmatrix}
\end{aligned}$$

□

Finally, we are ready to prove the law, $(\mu\, P \bullet c \rightarrow P) \setminus \{c\} = \mathit{Chaos}$, and the proof is simply the combination of the laws and theorems above.

Theorem 30.

$$(\mu\, P \bullet c \rightarrow P) \setminus \{c\} = \mathit{Chaos}$$

Proof.

$$
\begin{array}{ll}
& (\mu\, P \bullet c \to P) \setminus \{c\} & [\text{Theorem 17}] \\
= & \mathbf{R_{ct}}(\neg\, ((\exists\, s, r \bullet (\mu\, P \bullet c \to P)^f_f[s, r/tr', ref'] \wedge L_t);\ \mathbf{R1_{ct}}(\mathbf{true})) \vdash EE^{7}) & [\text{def of } \vdash] \\
= & \mathbf{R_{ct}}\left(\ ((\exists\, s, r \bullet (\mu\, P \bullet c \to P)^f_f[s, r/tr', ref'] \wedge L_t);\ \mathbf{R1_{ct}}(\mathbf{true})) \vee \neg\ ok \vee (ok' \wedge EE)\ \right) & \\
& & [\text{Theorem 18 and Law 82}] \\
= & \mathbf{R_{ct}}\left(\left(\left(\exists\, s,r \bullet \left(\neg \nu \left(\lambda\, X \bullet \neg\left(\left(\begin{array}{c} term\_now\_com(c) \vee \\ terminating\_com(c) \end{array}\right); \mathbf{R1_{ct}}\left(\begin{array}{c} \neg\ wait \wedge \\ \mathbf{R2_{ct}}(\neg\, X) \end{array}\right)\right)\right)\right)[s,r/tr',ref'] \wedge L_t\right); \mathbf{R1_{ct}}(\mathbf{true})\right)\right) & \\
& \qquad \vee \neg\ ok \vee (ok' \wedge EE) & \\
& & [\text{Lemma 38}] \\
= & \mathbf{R_{ct}}\left(\left(\left(\exists s, r \bullet \left(\begin{array}{l} \prod_{n=0}^{\infty} \left(\left(\begin{array}{c} front(ref') = front(ref) \wedge \\ diff(tr', tr) = \langle\langle c \rangle^n\rangle \wedge state' = state \end{array}\right); \mathbf{R1_{ct}}(\neg\, \mathbf{false})\right) \vee \\ \neg \prod_{n=0}^{\infty} \neg \left(\left(\begin{array}{c} term\_now\_com(c) \vee \\ terminating\_com(c) \end{array}\right); \mathbf{R1_{ct}}(\neg\, wait \wedge \mathbf{R2_{ct}}(\neg \mathbf{false}))\right) \end{array}\right)^n [s,r/tr',ref'] \wedge L_t\right); \right.\right. & \\
& \qquad \left.\left. \mathbf{R1_{ct}}(\mathbf{true}) \right.\right) & \\
& \qquad \vee \neg\ ok \vee (ok' \wedge EE)\Big) & \\
& & [\text{property of } \textstyle\prod \text{ and predicate calculus}] \\
= & \mathbf{R_{ct}}\left(\left(\left(\begin{array}{l} \exists s, r \bullet \left(\prod_{n=0}^{\infty} \left(\left(\begin{array}{c} front(ref') = front(ref) \wedge \\ diff(tr', tr) = \langle\langle c \rangle^n\rangle \wedge state' = state \end{array}\right); \mathbf{R1_{ct}}(\neg\, \mathbf{false})\right)\right)[s,r/tr',ref'] \wedge L_t \\ \qquad \vee \\ \exists s, r \bullet \left(\neg \prod_{n=0}^{\infty} \neg \left(\left(\begin{array}{c} term\_now\_com(c) \vee \\ terminating\_com(c) \end{array}\right); \mathbf{R1_{ct}}(\neg\, wait \wedge \mathbf{R2_{ct}}(\neg \mathbf{false}))\right)^n\right)[s,r/tr',ref'] \wedge L_t \end{array}\right); \right. & \\
& \qquad \left. \mathbf{R1_{ct}}(\mathbf{true}) \right. & \\
& \qquad \vee \neg\ ok \vee (ok' \wedge EE)\Big) & \\
& & [L_t(4.78), \text{ property of } \textstyle\prod \text{ and relational calculus}] \\
= & \mathbf{R_{ct}}\left(\left(\left(\begin{array}{c} ((tr' = tr \wedge front(ref') = front(ref));\ \mathbf{R1_{ct}}(\mathbf{true})) \vee \\ \left(\exists s, r \bullet \left(\neg \prod_{n=0}^{\infty} \neg \left(\left(\begin{array}{c} term\_now\_com(c) \vee \\ terminating\_com(c) \end{array}\right); \mathbf{R1_{ct}}(\neg\, wait \wedge \mathbf{R2_{ct}}(\neg \mathbf{false}))\right)^n\right)[s,r/tr',ref'] \wedge L_t\right); \right. \\ \left. \mathbf{R1_{ct}}(\mathbf{true}) \right. \end{array}\right) & \\
& \qquad \vee \neg\ ok \vee (ok' \wedge EE)\Big) & \\
& & [\mathbf{R1_{ct}} \text{ and relational calculus}] \\
= & \mathbf{R_{ct}}\left(\left(\left(\begin{array}{c} \mathbf{true} \vee \\ \left(\exists s, r \bullet \left(\neg \prod_{n=0}^{\infty} \neg \left(\left(\begin{array}{c} term\_now\_com(c) \vee \\ terminating\_com(c) \end{array}\right); \mathbf{R1_{ct}}(\neg\, wait \wedge \mathbf{R2_{ct}}(\neg \mathbf{false}))\right)^n\right)[s,r/tr',ref'] \wedge L_t\right); \right. \\ \left. \mathbf{R1_{ct}}(\mathbf{true}) \right. \end{array}\right) & \\
& \qquad \vee \neg\ ok \vee (ok' \wedge EE)\Big) & \\
& & [\text{propositional calculus}] \\
= & \mathbf{R_{ct}}(\mathbf{true}) & [Chaos] \\
= & Chaos &
\end{array}
$$

□

In addition, more cases of the relation between *Miracle* and other operators have been explored in [23]. For example, if we put *Miracle* and an ordinary action in parallel, the result is *Miracle* itself deduced from our intuitive conclusion that all processes participating in a parallel must start at the same time, but *Miracle* can never start, and thereby the whole process can never start either. The proof of this subtle process can be found in [23].

[7] We simply use *EE* to denote the unused part of the proof.

6 Conclusion

In this report we have developed a new *Circus Time* model which is an extension to the original *Circus Time* model. The semantics of the new model is also based on UTP, providing a wider variety of timing operators which is considered sufficient for expressing and reasoning about timing behaviour of SCJ programs. In addition, each action in *Circus Time* is expressed as a reactive design. Compared with the relational UTP semantics in the old model, the reactive-design semantics is a more concise, readable and uniform UTP semantics. Although this semantics has been explored in the early work [3, 13, 22], we here provide a complete version. That is, we provide a reactive design for each operator in *Circus Time*.

The deadline operators, one of major contributions in this model, is extremely helpful when reasoning about SCJ programs. It is similar to operators in existing process algebras and temporal logic [7, 19, 16]. We adopt a similar idea to that in [7]: failure to meet a deadline results in infeasibility in the model of the program. A deadline is viewed as an assertion for static analysis [6, 15]; it is used to identify timing paths and to inform a schedulability analysis. If the deadline cannot be guaranteed, the program is rejected. In *Circus Time* a deadline are treated as a requirement to a program's environment, imposing its environment to interact. Failure to meet the deadline will lead to a miraculous behaviour. In this model, the deadline operators are defined by means of a brand-new action *Miracle* which denotes an unstarted state. In fact, this action has always been remained in the semantics of the original *Circus Time*, but has never been fully explored until now. Moreover, we have investigated any combination of *Miracle* with other operator, which is vital for developing the right operational semantics.

In the future work we will mechanise the *Circus Time* model in a theorem prover since handwritten proof is error-prone. Oliveira [14] has embedded the denotational semantics of *Circus* in the theorem prover ProofPower and later Zeyda [29] has complemented this embedding by extending it to deal with general UTP theories. We will use their approaches to mechanise *Circus Time*.

A Properties of healthiness conditions

A.1 Property of sequences

Property 5. *Suppose all traces such as s and t are non-empty.*

$L1.\ s = t \Rightarrow s \preccurlyeq t$

$L2.\ s = t \Rightarrow front(s) \leq t$

$L3.\ (\exists\, r \bullet front(s) \preccurlyeq r \land front(r) \preccurlyeq t) = s \preccurlyeq t$

$L4.\ (\exists\, r \bullet s \preccurlyeq r \land r \preccurlyeq t) = s \preccurlyeq t$

$L5.\ (s \preccurlyeq r \land r \preccurlyeq t) \Rightarrow (\#diff(r, s) + \#diff(t, r)) = \#diff(t, s)$

$L6.\ \#tr = \#ref \land \#tr' = \#ref' \Rightarrow \#ref = \langle\langle\rangle\rangle \land \#diff(tr', tr) = \#ref'$

$L7.\ diff(t, \langle\langle\rangle\rangle) = t$

$L8.\ diff(t, t) = \langle\langle\rangle\rangle$

$L9.\ diff(diff(t, s), r) = diff(t, conc(s, r))$

$L10.\ s \preccurlyeq t \Rightarrow (diff(t, s) = r) = (conc(s, r) = t)$

$L11.\ s \preccurlyeq t = \langle\langle\rangle\rangle \preccurlyeq diff(t, s)$

$L12.\ head(t - front(s)) = t(\#s)$

$L13.\ \langle t(\#s)\rangle \frown tail(t - front(s)) = t - front(s)$

$L14.\ (\frown/t - \frown/s = \langle a\rangle \land \#t = \#s) = (diff(t, s) = \langle\langle a\rangle\rangle)$

A.2 Property of $\mathbf{R1_{ct}}$

Law 2. $\mathbf{R1_{ct}} \circ \mathbf{R1_{ct}}(P) = \mathbf{R1_{ct}}(P)$

Proof.

$$
\begin{aligned}
&\mathbf{R1_{ct}} \circ \mathbf{R1_{ct}}(P) && [\mathbf{R1_{ct}}]\\
=&P \wedge RT \wedge RT && [RT \text{ is idempotent}]\\
=&P \wedge RT && [\mathbf{R1_{ct}}]\\
=&\mathbf{R1_{ct}}(P)
\end{aligned}
$$

□

Law 3. $\mathbf{R1_{ct}}(P \wedge Q) = \mathbf{R1_{ct}}(P) \wedge \mathbf{R1_{ct}}(Q)$

Proof.

$$
\begin{aligned}
&\mathbf{R1_{ct}}(P \wedge Q) && [\mathbf{R1_{ct}}]\\
=&P \wedge Q \wedge RT && [\text{propositional calculus}]\\
=&P \wedge RT \wedge Q \wedge RT && [\mathbf{R1_{ct}}]\\
=&\mathbf{R1_{ct}}(P) \wedge \mathbf{R1_{ct}}(Q)
\end{aligned}
$$

□

Law 4. $\mathbf{R1_{ct}}(P \vee Q) = \mathbf{R1_{ct}}(P) \vee \mathbf{R1_{ct}}(Q)$

Proof.

$$
\begin{aligned}
&\mathbf{R1_{ct}}(P \vee Q) && [\mathbf{R1_{ct}}]\\
=&(P \vee Q) \wedge RT && [\text{propositional calculus}]\\
=&(P \wedge RT) \vee (Q \wedge RT) && [\mathbf{R1_{ct}}]\\
=&\mathbf{R1_{ct}}(P) \vee \mathbf{R1_{ct}}(Q)
\end{aligned}
$$

□

Law 5. $\mathbf{R1_{ct}}(\mathbb{I}_{ct}) = \mathbb{I}_{ct}$

Proof.

$$
\begin{aligned}
&\mathbf{R1_{ct}}(\mathbb{I}_{ct}) && [\mathbf{R1_{ct}}]\\
=&\mathbb{I}_{ct} \wedge RT && [\mathbb{I}_{ct}]\\
=&((\neg\ ok \wedge RT) \vee (ok' \wedge \mathbb{I})) \wedge RT && [\text{propositional calculus}]\\
=&(\neg\ ok \wedge RT) \vee (ok' \wedge \mathbb{I} \wedge RT)\\
=&(\neg\ ok \wedge RT) \vee && [\text{property 5-L1,L2}]\\
&(ok' \wedge \mathbb{I} \wedge tr' = tr \wedge ref' = ref \wedge tr \preccurlyeq tr' \wedge front(ref) \leq ref' \wedge \#tr = \#ref \wedge \#tr' = \#ref')\\
=&(\neg\ ok \wedge RT) \vee (ok' \wedge \mathbb{I} \wedge tr' = tr \wedge ref' = ref \wedge \#tr = \#ref \wedge \#tr' = \#ref') && [\mathbb{I}_{ct}]\\
=&\mathbb{I}_{ct}
\end{aligned}
$$

□

Law 6. $\mathbf{R1_{ct}}(P;\ Q) = P;\ Q$ provided P and Q are $\mathbf{R1_{ct}}$ healthy

Proof.

$$
\begin{array}{ll}
& P;\ Q & \text{[assumption]}\\
= & \mathbf{R1_{ct}}(P);\ \mathbf{R1_{ct}}(Q) & [\mathbf{R1_{ct}}]\\
= & \left(\begin{array}{l}(P \wedge tr \preccurlyeq tr' \wedge front(ref) \leq ref' \wedge \#tr = \#ref \wedge \#tr' = \#ref');\\ (Q \wedge tr \preccurlyeq tr' \wedge front(ref) \leq ref' \wedge \#tr = \#ref \wedge \#tr' = \#ref')\end{array}\right) & \text{[sequential composition]}\\
= & \exists\, tr_0, ref_0, v_0 \bullet \left(\begin{array}{l}P[tr_0, ref_0, v_0/tr', ref', v'] \wedge tr \preccurlyeq tr_0 \wedge front(ref) \leq ref_0 \wedge \#tr = \#ref \wedge \#tr_0 = \#ref_0 \wedge\\ Q[tr_0, ref_0, v_0/tr, ref, v] \wedge tr_0 \preccurlyeq tr' \wedge front(ref_0) \leq ref' \wedge \#tr_0 = \#ref_0 \wedge \#tr' = \#ref'\end{array}\right) & \text{[property 5-L3,L4]}\\
= & \exists\, tr_0, ref_0, v_0 \bullet (P[tr_0, ref_0, v_0/tr', ref', v'] \wedge Q[tr_0, ref_0, v_0/tr, ref, v]) \wedge & [\mathbf{R1_{ct}}]\\
& \quad tr \preccurlyeq tr' \wedge front(ref) \leq ref' \wedge \#tr = \#ref \wedge \#tr' = \#ref'\\
= & \mathbf{R1_{ct}}(P;\ Q)
\end{array}
$$

□

Law 7. $\mathbf{R1_{ct}}(P \lhd b \rhd Q) = \mathbf{R1_{ct}}(P) \lhd b \rhd \mathbf{R1_{ct}}(Q)$

Proof.

$$
\begin{array}{ll}
& \mathbf{R1_{ct}}(P \lhd b \rhd Q) & [\lhd\rhd \text{ and } \mathbf{R1_{ct}}]\\
= & ((b \wedge P) \vee (\neg\, b \wedge Q)) \wedge RT & \text{[propositional calculus]}\\
= & (b \wedge P \wedge RT) \vee (\neg\, b \wedge Q \wedge RT) & [\lhd\rhd \text{ and } \mathbf{R1_{ct}}]\\
= & \mathbf{R1_{ct}}(P) \lhd b \rhd \mathbf{R1_{ct}}(Q)
\end{array}
$$

□

A.3 Property of $\mathbf{R2_{ct}}$

Law 8. $\mathbf{R2_{ct}} \circ \mathbf{R2_{ct}}(P) = \mathbf{R2_{ct}}(P)$

Proof.

$$
\begin{array}{ll}
& \mathbf{R2_{ct}} \circ \mathbf{R2_{ct}}(P) & [\mathbf{R2_{ct}}]\\
= & P(\langle\langle\rangle\rangle, diff(tr', tr))(\langle\langle\rangle\rangle, diff(tr', tr)) & \text{[substitution]}\\
= & P(\langle\langle\rangle\rangle, diff(diff(tr', tr), \langle\langle\rangle\rangle)) & \text{[property 5-L7]}\\
= & P(\langle\langle\rangle\rangle, diff(tr', tr)) & [\mathbf{R2_{ct}}]\\
= & \mathbf{R2_{ct}}(P)
\end{array}
$$

□

Law 9. $\mathbf{R2_{ct}}(P \wedge Q) = \mathbf{R2_{ct}}(P) \wedge \mathbf{R2_{ct}}(Q)$

Proof.

$$
\begin{array}{ll}
& \mathbf{R2_{ct}}(P \wedge Q) & [\mathbf{R2_{ct}}]\\
= & (P \wedge Q)(\langle\langle\rangle\rangle, diff(tr', tr)) & \text{[substitution]}\\
= & P(\langle\langle\rangle\rangle, diff(tr', tr)) \wedge Q(\langle\langle\rangle\rangle, diff(tr', tr)) & [\mathbf{R2_{ct}}]\\
= & \mathbf{R2_{ct}}(P) \wedge \mathbf{R2_{ct}}(Q)
\end{array}
$$

□

Law 10. $\mathbf{R2_{ct}}(P \vee Q) = \mathbf{R2_{ct}}(P) \vee \mathbf{R2_{ct}}(Q)$

Proof.

$$
\begin{aligned}
&\mathbf{R2_{ct}}(P \vee Q) && [\mathbf{R2_{ct}}]\\
=&(P \vee Q)(\langle\langle\rangle\rangle, \mathit{diff}(tr', tr)) && [\text{substitution}]\\
=&P(\langle\langle\rangle\rangle, \mathit{diff}(tr', tr)) \vee Q(\langle\langle\rangle\rangle, \mathit{diff}(tr', tr)) && [\mathbf{R2_{ct}}]\\
=&\mathbf{R2_{ct}}(P) \vee \mathbf{R2_{ct}}(Q)
\end{aligned}
$$

□

Law 11. $\mathbf{R2_{ct}}(\mathbb{I}_{ct}) = \mathbb{I}_{ct}$

Proof.

$$
\begin{aligned}
&\mathbf{R2_{ct}}(\mathbb{I}_{ct})\\
=&\left(\left(\begin{array}{l}(\neg\ ok \wedge tr \preccurlyeq tr' \wedge \mathit{front}(ref) \leq ref'\\ \quad \wedge\ \#tr = \#ref \wedge \#tr' = \#ref'\end{array}\right) \vee (ok' \wedge \mathbb{I} \wedge \#tr = \#ref \wedge \#tr' = \#ref')\right)(\langle\langle\rangle\rangle, \mathit{diff}(tr', tr))\\
& && [\text{substitution}]\\
=&(\neg\ ok \wedge \langle\langle\rangle\rangle \preccurlyeq \mathit{diff}(tr', tr) \wedge \mathit{front}(ref) \leq ref' \wedge \#\langle\langle\rangle\rangle = \#ref \wedge \#\mathit{diff}(tr', tr) = \#ref')\\
&\vee (ok' \wedge \mathbb{I} \wedge \mathit{diff}(tr', tr) = \langle\langle\rangle\rangle \wedge \#\langle\langle\rangle\rangle = \#ref \wedge \#\mathit{diff}(tr', tr) = \#ref') && [\text{property 5-L8,L11}]\\
=&(\neg\ ok \wedge tr \preccurlyeq tr' \wedge \mathit{front}(ref) \leq ref' \wedge \#\langle\langle\rangle\rangle = \#ref \wedge \#\mathit{diff}(tr', tr) = \#ref') \vee\\
&(ok' \wedge \mathbb{I} \wedge tr' = tr \wedge \#\langle\langle\rangle\rangle = \#ref \wedge \#\mathit{diff}(tr', tr) = \#ref') && [\text{property 5-L6}]\\
\Leftarrow&(\neg\ ok \wedge tr \preccurlyeq tr' \wedge \mathit{front}(ref) \leq ref' \wedge \#tr = \#ref \wedge \#tr' = \#ref') \vee\\
&(ok' \wedge \mathbb{I} \wedge tr' = tr \wedge \#tr = \#ref \wedge \#tr' = \#ref') && [\mathbb{I}_{ct}]\\
=&\mathbb{I}_{ct}
\end{aligned}
$$

□

Lemma 39. *For a fresh variables u,* $\mathbf{R2_{ct}}(P\,;\,Q) = (P(\langle\langle\rangle\rangle, tr')\,;\,Q(tr, \mathit{diff}(tr', u))[tr/u]$

Proof.

$$
\begin{aligned}
&\mathbf{R2_{ct}}(P\,;\,Q) && [\mathbf{R2_{ct}}]\\
=&(P\,;\,Q)(\langle\langle\rangle\rangle, \mathit{diff}(tr', tr)) && [\text{substitution}]\\
=&((P\,;\,Q)(\langle\langle\rangle\rangle, \mathit{diff}(tr', u)))[tr/u] && [\text{sequence substitution}]\\
=&(P(\langle\langle\rangle\rangle, tr')\,;\,Q(tr, \mathit{diff}(tr', u)))[tr/u]
\end{aligned}
$$

□

Law 12. $\mathbf{R2_{ct}}(P;\,Q) = P;\,Q$ provided P and Q are $\mathbf{R2_{ct}}$ healthy

Proof.

$$
\begin{aligned}
&\mathbf{R2_{ct}}(P\,;\,Q) && [\text{Lemma 39}]\\
=&(P(\langle\langle\rangle\rangle, tr')\,;\,Q(tr, \mathit{diff}(tr', u)))[tr/u] && [Q \text{ is } \mathbf{R2_{ct}}]\\
=&(P(\langle\langle\rangle\rangle, tr')\,;\,Q(\langle\langle\rangle\rangle, \mathit{diff}(\mathit{diff}(tr', u), tr)))[tr/u] && [\text{property 5-L9}]\\
=&(P(\langle\langle\rangle\rangle, tr')\,;\,Q(\langle\langle\rangle\rangle, \mathit{diff}(tr', \mathit{conc}(u, tr))))[tr/u] && [[\text{property of assign (leading assign)}]]\\
=&(P(\langle\langle\rangle\rangle, tr')\,;\,tr := \mathit{conc}(u, tr);\,Q(\langle\langle\rangle\rangle, \mathit{diff}(tr', tr)))[tr/u]\\
& && [[\text{property of assign(following assign) and property 5-L10}]\\
=&(P(\langle\langle\rangle\rangle, \mathit{diff}(tr', u))\,;\,Q(\langle\langle\rangle\rangle, \mathit{diff}(tr', tr)))[tr/u] && [\text{substitution}]\\
=&(P(\langle\langle\rangle\rangle, \mathit{diff}(tr', tr))\,;\,Q(\langle\langle\rangle\rangle, \mathit{diff}(tr', tr))) = && P;\,Q \quad [P \text{ and } Q \text{ are } \mathbf{R2_{ct}}]
\end{aligned}
$$

□

Law 13. $\mathbf{R2_{ct}}(P \lhd b \rhd Q) = \mathbf{R2_{ct}}(P) \lhd b \rhd \mathbf{R2_{ct}}(Q)$ if b not contain tr and tr'

Proof.

$$
\begin{aligned}
&\mathbf{R2_{ct}}(P \lhd b \rhd Q) && [\lhd\rhd \text{ and } \mathbf{R2_{ct}}]\\
=&((b \wedge P) \vee (\neg\, b \wedge Q))(\langle\langle\rangle\rangle, \mathit{diff}(tr', tr)) && [\text{assumption and substitution}]\\
=&(b \wedge P(\langle\langle\rangle\rangle, \mathit{diff}(tr', tr))) \vee (\neg\, b \wedge Q(\langle\langle\rangle\rangle, \mathit{diff}(tr', tr))) && [\mathbf{R2_{ct}}]\\
=&\mathbf{R2_{ct}}(P) \lhd b \rhd \mathbf{R2_{ct}}(Q)
\end{aligned}
$$

□

Law 14. $\mathbf{R2_{ct}} \circ \mathbf{R1_{ct}} = \mathbf{R1_{ct}} \circ \mathbf{R2_{ct}}$

Proof.

$$
\begin{aligned}
&\mathbf{R2_{ct}} \circ \mathbf{R1_{ct}}(P) && [\mathbf{R1_{ct}}]\\
=&\mathbf{R2_{ct}}(P \wedge tr \preccurlyeq tr' \wedge \mathit{front}(ref) \leq ref' \wedge \#tr = \#ref \wedge \#tr' = \#ref') && [\mathbf{R2_{ct}}]\\
=&(P \wedge tr \preccurlyeq tr' \wedge \mathit{front}(ref) \leq ref' \wedge \#tr = \#ref \wedge \#tr' = \#ref')(\langle\langle\rangle\rangle, \mathit{diff}(tr', tr)) && [\text{subs.}]\\
=&P(\langle\langle\rangle\rangle, \mathit{diff}(tr', tr)) \wedge \langle\langle\rangle\rangle \preccurlyeq \mathit{diff}(tr', tr) \wedge \mathit{front}(ref) \leq ref' \wedge && [\text{property 5-L6,L11}]\\
&\#\langle\langle\rangle\rangle = \#ref \wedge \#\mathit{diff}(tr', tr) = \#ref'\\
=&P(\langle\langle\rangle\rangle, \mathit{diff}(tr', tr)) \wedge tr \preccurlyeq tr' \wedge \mathit{front}(ref) \leq ref' \wedge \#tr = \#ref \wedge \#tr' = \#ref' && [\mathbf{R2_{ct}}]\\
=&\mathbf{R2_{ct}}(P) \wedge tr \preccurlyeq tr' \wedge \mathit{front}(ref) \leq ref' \wedge \#tr = \#ref \wedge \#tr' = \#ref' && [\mathbf{R1_{ct}}]\\
=&\mathbf{R1_{ct}} \circ \mathbf{R2_{ct}}(P)
\end{aligned}
$$

□

A.4 Property of $\mathbf{R3_{ct}}$

Law 15. $\mathbf{R3_{ct}} \circ \mathbf{R3_{ct}}(P) = \mathbf{R3_{ct}}(P)$

Proof.

$$
\begin{aligned}
&\mathbf{R3_{ct}} \circ \mathbf{R3_{ct}}(P) && [\mathbf{R3_{ct}}]\\
=&\mathbb{I}_{ct} \lhd \mathit{wait} \rhd (\mathbb{I}_{ct} \lhd \mathit{wait} \rhd P) && [\text{property of } \lhd\rhd]\\
=&\mathbb{I}_{ct} \lhd \mathit{wait} \rhd P && [\mathbf{R3_{ct}}]\\
=&\mathbf{R3_{ct}}(P)
\end{aligned}
$$

□

Law 16. $\mathbf{R3_{ct}}(P \wedge Q) = \mathbf{R3_{ct}}(P) \wedge \mathbf{R3_{ct}}(Q)$

Proof.

$$
\begin{aligned}
&\mathbf{R3_{ct}}(P \wedge Q) && [\mathbf{R3_{ct}}]\\
=&\mathbb{I}_{ct} \lhd \mathit{wait} \rhd (P \wedge Q) && [\lhd\rhd]\\
=&(\mathbb{I}_{ct} \wedge \mathit{wait}) \vee (\neg\, \mathit{wait} \wedge P \wedge Q) && [\text{propositional calculus}]\\
=&((\mathbb{I}_{ct} \wedge \mathit{wait}) \vee (\neg\, \mathit{wait} \wedge P)) \wedge ((\mathbb{I}_{ct} \wedge \mathit{wait}) \vee (\neg\, \mathit{wait} \wedge Q)) && [\mathbf{R3_{ct}}]\\
=&\mathbf{R3_{ct}}(P) \wedge \mathbf{R3_{ct}}(Q)
\end{aligned}
$$

□

Law 17. $\mathbf{R3_{ct}}(P \vee Q) = \mathbf{R3_{ct}}(P) \vee \mathbf{R3_{ct}}(Q)$

Proof.

$$
\begin{aligned}
&\mathbf{R3_{ct}}(P \vee Q) && [\mathbf{R3_{ct}}]\\
=&\mathbb{I}_{ct} \lhd \mathit{wait} \rhd (P \vee Q) && [\lhd\rhd]\\
=&(\mathbb{I}_{ct} \wedge \mathit{wait}) \vee (\neg\, \mathit{wait} \wedge (P \vee Q)) && [\text{propositional calculus}]\\
=&(\mathbb{I}_{ct} \wedge \mathit{wait}) \vee (\neg\, \mathit{wait} \wedge P) \vee (\neg\, \mathit{wait} \wedge Q) && [\mathbf{R3_{ct}}]\\
=&\mathbf{R3_{ct}}(P) \vee \mathbf{R3_{ct}}(Q)
\end{aligned}
$$

□

Law 18. $\mathbf{R3_{ct}}(\mathbb{I}_{ct}) = \mathbb{I}_{ct}$

Proof.

$$
\begin{aligned}
&\mathbf{R3_{ct}}(\mathbb{I}_{ct}) && [\mathbf{R3_{ct}}]\\
=&\mathbb{I}_{ct} \lhd \mathit{wait} \rhd \mathbb{I}_{ct} && [\text{property of } \lhd\rhd]\\
=&\mathbb{I}_{ct}
\end{aligned}
$$

□

Lemma 40. $RT;\ RT = RT$

Lemma 41. $RT(tr, tr') = RT(\langle\langle\rangle\rangle, \mathit{diff}(tr'tr))$

Lemma 42. $RT;\ P = RT$ *if* P *is* $\mathbf{R1_{ct}}$ *and* $\mathbf{R3_{ct}}$

Proof.

$$
\begin{aligned}
&RT;\ P && [\text{propositional calculus}]\\
=&RT;\ ((\mathit{ok} \vee \neg\ \mathit{ok}) \wedge (\mathit{wait} \vee \neg\ \mathit{wait}) \wedge P) && [\text{relational calculus}]\\
=&(RT;\ (\mathit{ok} \wedge \mathit{wait} \wedge P)) \vee (RT;\ (\mathit{ok} \wedge \neg\ \mathit{wait} \wedge P)) \vee && [P \text{ is } \mathbf{R3_{ct}} \text{ and propositional calculus}]\\
&(RT;\ (\neg\ \mathit{ok} \wedge \mathit{wait} \wedge P)) \vee (RT;\ (\neg\ \mathit{ok} \wedge \neg\ \mathit{wait} \wedge P))\\
=&(RT;\ (\mathit{ok} \wedge \mathit{wait} \wedge P)) \vee (RT;\ (\mathit{ok} \wedge \neg\ \mathit{wait} \wedge P)) \vee && [\mathbb{I}_{ct}]\\
&(RT;\ (\neg\ \mathit{ok} \wedge \mathit{wait} \wedge \mathbb{I}_{ct})) \vee (RT;\ (\neg\ \mathit{ok} \wedge \neg\ \mathit{wait} \wedge P))\\
=&(RT;\ (\mathit{ok} \wedge \mathit{wait} \wedge P)) \vee (RT;\ (\mathit{ok} \wedge \neg\ \mathit{wait} \wedge P)) \vee && [\text{Lemma 40}]\\
&(RT;\ (\neg\ \mathit{ok} \wedge \mathit{wait} \wedge RT)) \vee (RT;\ (\neg\ \mathit{ok} \wedge \neg\ \mathit{wait} \wedge P))\\
=&(RT;\ (\mathit{ok} \wedge \mathit{wait} \wedge P)) \vee (RT;\ (\mathit{ok} \wedge \neg\ \mathit{wait} \wedge P)) \vee RT \vee (RT;\ (\neg\ \mathit{ok} \wedge \neg\ \mathit{wait} \wedge P)) && [\text{Law 6}]\\
=&(RT;\ (\mathit{ok} \wedge \mathit{wait} \wedge P) \wedge RT) \vee (RT;\ (\mathit{ok} \wedge \neg\ \mathit{wait} \wedge P) \wedge RT) \vee RT \vee && [\text{propositinal calculus}]\\
&(RT;\ (\neg\ \mathit{ok} \wedge \neg\ \mathit{wait} \wedge P) \wedge RT)\\
=&RT
\end{aligned}
$$

□

Law 19. $\mathbf{R3_{ct}}(P;\ Q) = P;\ Q$ provided P is $\mathbf{R3_{ct}}$ and Q is $\mathbf{R1_{ct}}$ and $\mathbf{R3_{ct}}$

Proof.

$$
\begin{aligned}
&P;\ Q && [P \text{ is } \mathbf{R1_{ct}}]\\
=&(\mathbb{I}_{ct} \lhd \mathit{wait} \rhd P);\ Q && [\text{relational calculus and } \mathbb{I}_{ct}]\\
=&((\neg\ \mathit{ok} \wedge RT \wedge \mathit{wait});\ Q) \vee ((\mathit{ok}' \wedge \mathit{wait} \wedge \mathbb{I});\ Q) \vee ((\neg\ \mathit{wait} \wedge P);\ Q) && [\text{Lemma 40}]\\
=&(\neg\ \mathit{ok} \wedge RT \wedge \mathit{wait}) \vee ((\mathit{ok}' \wedge \mathit{wait} \wedge \mathbb{I});\ Q) \vee ((\neg\ \mathit{wait} \wedge P);\ Q) && [Q \text{ is } \mathbf{R3_{ct}}]\\
=&(\neg\ \mathit{ok} \wedge RT \wedge \mathit{wait}) \vee ((\mathit{ok}' \wedge \mathit{wait} \wedge \mathbb{I});\ (\mathbb{I}_{ct} \wedge \mathit{wait})) \vee ((\neg\ \mathit{wait} \wedge P);\ Q) && [\text{rel. calculus and } \mathbb{I}_{ct}]\\
=&(\mathbb{I}_{ct} \wedge \mathit{wait}) \vee ((\neg\ \mathit{wait} \wedge P);\ Q) && [\text{relational calculus}]\\
=&(\mathbb{I}_{ct} \wedge \mathit{wait}) \vee (\neg\ \mathit{wait} \wedge (P;\ Q)) && [\mathbf{R3_{ct}}]\\
=&\mathbf{R3_{ct}}(P;\ Q)
\end{aligned}
$$

□

Law 20. $\mathbf{R3_{ct}}(P \lhd b \rhd Q) = \mathbf{R3_{ct}}(P) \lhd b \rhd \mathbf{R3_{ct}}(Q)$

Proof.

$$
\begin{aligned}
&\mathbf{R3_{ct}}(P \lhd b \rhd Q) && [\mathbf{R3_{ct}} \text{ and } \lhd\rhd]\\
=&(\mathbb{I}_{ct} \wedge \mathit{wait}) \vee (\neg\ \mathit{wait} \wedge P \wedge b) \vee (\neg\ \mathit{wait} \wedge Q \wedge \neg\ b) && [\text{propositional calculus}]\\
=&(\mathbb{I}_{ct} \wedge \mathit{wait} \wedge (b \vee \neg\ b)) \vee (\neg\ \mathit{wait} \wedge P \wedge b) \vee (\neg\ \mathit{wait} \wedge Q \wedge \neg\ b) && [\mathbf{R3_{ct}}]\\
=&\mathbf{R3_{ct}}(P) \lhd b \rhd \mathbf{R3_{ct}}(Q)
\end{aligned}
$$

□

Law 21. $\mathbf{R3_{ct}} \circ \mathbf{R1_{ct}} = \mathbf{R1_{ct}} \circ \mathbf{R3_{ct}}$

Proof.

$$
\begin{array}{ll}
& \mathbf{R3_{ct}} \circ \mathbf{R1_{ct}}(P) & [\mathbf{R3_{ct}}\ ] \\
= & \mathit{II}_{ct} \lhd \mathit{wait} \rhd \mathbf{R1_{ct}}(P) & [\text{Law 5}] \\
= & \mathbf{R1_{ct}}(\mathit{II}_{ct}) \lhd \mathit{wait} \rhd \mathbf{R1_{ct}}(P) & [\text{Law 7}] \\
= & \mathbf{R1_{ct}}(\mathit{II}_{ct} \lhd \mathit{wait} \rhd P) & [\mathbf{R3_{ct}}] \\
= & \mathbf{R1_{ct}} \circ \mathbf{R3_{ct}}(P)
\end{array}
$$

□

Law 22. $\mathbf{R3_{ct}} \circ \mathbf{R2_{ct}} = \mathbf{R2_{ct}} \circ \mathbf{R3_{ct}}$

Proof.

$$
\begin{array}{ll}
& \mathbf{R3_{ct}} \circ \mathbf{R2_{ct}}(P) & [\mathbf{R3_{ct}}] \\
= & \mathit{II}_{ct} \lhd \mathit{wait} \rhd \mathbf{R2_{ct}}(P) & [\text{Law 11}] \\
= & \mathbf{R2_{ct}}(\mathit{II}_{ct}) \lhd \mathit{wait} \rhd \mathbf{R2_{ct}}(P) & [\text{Law 13}] \\
= & \mathbf{R2_{ct}}(\mathit{II}_{ct} \lhd \mathit{wait} \rhd P) & [\mathbf{R3_{ct}}] \\
= & \mathbf{R2_{ct}} \circ \mathbf{R3_{ct}}(P)
\end{array}
$$

□

Law 23. $(\mathbf{R3_{ct}}(P))^c = ((\mathit{II}_{ct})^c \lhd \mathit{wait} \rhd P^c)$

Proof.

$$
\begin{array}{ll}
& (\mathbf{R3_{ct}}(P))^c & [\mathbf{R3_{ct}}] \\
= & (\mathit{II}_{ct} \lhd \mathit{wait} \rhd P)^c & [\text{predicate calculus}] \\
= & (\mathit{II}_{ct})^c \lhd \mathit{wait} \rhd P^c
\end{array}
$$

□

Law 24. $(\mathbf{R_{ct}}(P))_f = \mathbf{R1_{ct}} \circ \mathbf{R2_{ct}}(P_f)$

Proof.

$$
\begin{array}{ll}
& (\mathbf{R_{ct}}(P))_f & [\mathbf{R_{ct}} \text{ and } \mathbf{R3_{ct}}] \\
= & (\mathit{II}_{ct} \lhd \mathit{wait} \rhd \mathbf{R1_{ct}} \circ \mathbf{R2_{ct}}(P))_f & [\text{substitution}] \\
= & (\mathbf{R1_{ct}} \circ \mathbf{R2_{ct}}(P))_f & [\text{predicate calculus}] \\
= & \mathbf{R1_{ct}} \circ \mathbf{R2_{ct}}(P_f)
\end{array}
$$

□

Law 25. $(\mathbf{R_{ct}}(P))_t = (\mathit{II}_{ct})_t$

Proof.

$$
\begin{array}{ll}
& (\mathbf{R_{ct}}(P))_t & [\mathbf{R_{ct}} \text{ and } \mathbf{R3_{ct}}] \\
= & (\mathit{II}_{ct} \lhd \mathit{wait} \rhd \mathbf{R1_{ct}} \circ \mathbf{R2_{ct}}(P))_t & [\text{substitution}] \\
= & (\mathit{II}_{ct})_t
\end{array}
$$

□

A.5 Property of $\mathbf{CSP1_{ct}}$

Law 26. $\mathbf{CSP1_{ct}} \circ \mathbf{CSP1_{ct}} = \mathbf{CSP1_{ct}}$

Proof.

$$
\begin{aligned}
&\mathbf{CSP1_{ct}} \circ \mathbf{CSP1_{ct}}(P) && [\mathbf{CSP1_{ct}}]\\
=&P \vee (\neg\ ok \wedge RT) \vee (\neg\ ok \wedge RT) && [\text{propositional calculus}]\\
=&P \vee (\neg\ ok \wedge RT) && [\mathbf{CSP1_{ct}}]\\
=&\mathbf{CSP1_{ct}}(P)
\end{aligned}
$$

□

Law 27. $\mathbf{CSP1_{ct}} \circ \mathbf{R1_{ct}} = \mathbf{R1_{ct}} \circ \mathbf{CSP1_{ct}}$

Proof.

$$
\begin{aligned}
&\mathbf{CSP1_{ct}} \circ \mathbf{R1_{ct}}(P) && [\mathbf{CSP1_{ct}} \text{ and } \mathbf{R1_{ct}}]\\
=&(P \wedge RT) \vee (\neg\ ok \wedge RT) && [\text{propositional calculus}]\\
=&(P \vee (\neg\ ok \wedge RT)) \wedge (RT \vee (\neg\ ok \wedge RT)) && [\mathbf{CSP1_{ct}} \text{ and propositional calculus}]\\
=&\mathbf{CSP1_{ct}}(P) \wedge RT && [\mathbf{R1_{ct}}]\\
=&\mathbf{R1_{ct}} \circ \mathbf{CSP1_{ct}}(P)
\end{aligned}
$$

□

Law 28. $\mathbf{CSP1_{ct}} \circ \mathbf{R2_{ct}} = \mathbf{R2_{ct}} \circ \mathbf{CSP1_{ct}}$

Proof.

$$
\begin{aligned}
&\mathbf{CSP1_{ct}} \circ \mathbf{R2_{ct}}(P) && [\mathbf{CSP1_{ct}} \text{ and } \mathbf{R2_{ct}}]\\
=&P(\langle\langle\rangle\rangle, \mathit{diff}(tr', tr)) \vee (\neg\ ok \wedge RT) && [\text{Lemma 41}]\\
=&P(\langle\langle\rangle\rangle, \mathit{diff}(tr', tr)) \vee (\neg\ ok \wedge RT(\langle\langle\rangle\rangle, \mathit{diff}(tr', tr))) && [\text{predicate calculus}]\\
=&(P \vee (\neg\ ok \wedge RT))(\langle\langle\rangle\rangle, \mathit{diff}(tr', tr)) && [\mathbf{CSP1_{ct}} \text{ and } \mathbf{R2_{ct}}]\\
=&\mathbf{R1_{ct}} \circ \mathbf{CSP1_{ct}}(P)
\end{aligned}
$$

□

Law 29. $\mathbf{CSP1_{ct}} \circ \mathbf{R3_{ct}} = \mathbf{R3_{ct}} \circ \mathbf{CSP1_{ct}}$

Proof.

$$
\begin{aligned}
&\mathbf{CSP1_{ct}} \circ \mathbf{R3_{ct}}(P) && [\mathbf{CSP1_{ct}} \text{ and } \mathbf{R3_{ct}}]\\
=&(\mathbb{I}_{ct} \lhd \mathit{wait} \rhd P) \vee (\neg\ ok \wedge RT) && [\lhd\rhd]\\
=&(\mathbb{I}_{ct} \wedge \mathit{wait}) \vee (\neg\ \mathit{wait} \wedge P) \vee (\neg\ ok \wedge RT) && [\text{propositional calculus}]\\
=&(\mathbb{I}_{ct} \wedge \mathit{wait}) \vee (\neg\ ok \wedge RT \wedge \mathit{wait}) \vee (\neg\ \mathit{wait} \wedge (P \vee (\neg\ ok \wedge RT))) && [\text{property of } \mathbb{I}_{ct}]\\
=&(\mathbb{I}_{ct} \wedge \mathit{wait}) \vee (\neg\ \mathit{wait} \wedge (P \vee (\neg\ ok \wedge RT))) && [\mathbf{CSP1_{ct}} \text{ and } \mathbf{R3_{ct}}]\\
=&\mathbf{R3_{ct}} \circ \mathbf{CSP1_{ct}}(P)
\end{aligned}
$$

□

Law 31. $\mathbf{CSP1_{ct}}(P) = \mathbf{R1_{ct}} \circ \mathbf{H1}(P)$ provided P is $\mathbf{R1_{ct}}$

Proof.

$$
\begin{aligned}
&\mathbf{CSP1_{ct}}(P) && [\mathbf{CSP1_{ct}}]\\
=&P \vee (\neg\ ok \wedge RT) && [P \text{ is } \mathbf{R1_{ct}}]\\
=&(P \wedge RT) \vee (\neg\ ok \wedge RT) && [\text{propositional calculus}]\\
=&RT \wedge (P \vee \neg\ ok) && [\mathbf{R1_{ct}} \text{ and } \mathbf{H1}]\\
=&\mathbf{R1_{ct}} \circ \mathbf{H1}(P)
\end{aligned}
$$

□

Law 32. $\mathbf{CSP1_{ct}}(P \wedge Q) = P \wedge Q$ provided P and Q are $\mathbf{CSP1_{ct}}$

Proof.

$$
\begin{array}{ll}
& \mathbf{CSP1_{ct}}(P \wedge Q) & [\mathbf{CSP1_{ct}}] \\
=& (P \wedge Q) \vee (\neg\ ok \wedge RT) & [\text{propositional calculus}] \\
=& (P \vee (\neg\ ok \wedge RT)) \wedge (Q \vee (\neg\ ok \wedge RT)) & [P \text{ and } Q \text{ are } \mathbf{CSP1_{ct}}] \\
=& P \wedge Q
\end{array}
$$

□

Law 33. $\mathbf{CSP1_{ct}}(P \vee Q) = P \vee Q$ provided P and Q are $\mathbf{CSP1_{ct}}$

Proof.

$$
\begin{array}{ll}
& \mathbf{CSP1_{ct}}(P \wedge Q) & [\mathbf{CSP1_{ct}}] \\
=& (P \vee Q) \vee (\neg\ ok \wedge RT) & [\text{propositional calculus}] \\
=& (P \vee (\neg\ ok \wedge RT)) \vee (Q \vee (\neg\ ok \wedge RT)) & [P \text{ and } Q \text{ are } \mathbf{CSP1_{ct}}] \\
=& P \vee Q
\end{array}
$$

□

Law 34. $\mathbf{CSP1_{ct}}(P \lhd b \rhd Q) = \mathbf{CSP1_{ct}}(P) \lhd b \rhd \mathbf{CSP1_{ct}}(Q)$

Proof.

$$
\begin{array}{ll}
& \mathbf{CSP1_{ct}}(P \lhd b \rhd Q) & [\mathbf{CSP1_{ct}} \text{ and } \lhd\rhd] \\
=& (P \wedge b) \vee (\neg\ b \wedge Q) \vee (\neg\ ok \wedge RT) & [\text{propositional calculus}] \\
=& (P \wedge b) \vee (\neg\ b \wedge Q) \vee (\neg\ ok \wedge RT \wedge (b \vee \neg\ b)) & [\text{propositional calculus}] \\
=& ((P \vee (\neg\ ok \wedge RT)) \wedge b) \vee ((Q \vee (\neg\ ok \wedge RT)) \wedge \neg\ b) & [\mathbf{CSP1_{ct}}] \\
=& (\mathbf{CSP1_{ct}}(P) \wedge b) \vee (\mathbf{CSP1_{ct}}(Q) \wedge \neg\ b) & [\lhd\rhd] \\
=& \mathbf{CSP1_{ct}}(P) \lhd b \rhd \mathbf{CSP1_{ct}}(Q)
\end{array}
$$

□

Law 35. $\mathbf{CSP1_{ct}}(P\,;\,Q) = P\,;\,Q$ provided P and Q are $\mathbf{CSP1_{ct}}$

Proof.

$$
\begin{array}{ll}
& \mathbf{CSP1_{ct}}(P\,;\,Q) & [P \text{ and } Q \text{ are } \mathbf{CSP1_{ct}}] \\
=& \mathbf{CSP1_{ct}}(\mathbf{CSP1_{ct}}(P); \mathbf{CSP1_{ct}}(Q)) & [\mathbf{CSP1_{ct}}] \\
=& (P;\ Q) \vee (P; (\neg\ ok \wedge RT)) \vee ((\neg\ ok \wedge RT);\ Q) \vee ((\neg\ ok \wedge RT); (\neg\ ok \wedge RT)) \vee (\neg\ ok \wedge RT) & [\text{relational calculus}] \\
=& (P;\ Q) \vee (P; (\neg\ ok \wedge RT)) \vee ((\neg\ ok \wedge RT);\ Q) \vee ((\neg\ ok \wedge RT); (\neg\ ok \wedge RT)) & [\text{relational calculus}] \\
=& (P \vee (\neg\ ok \wedge RT)); (Q \vee (\neg\ ok \wedge RT)) & [P \text{ and } Q \text{ are } \mathbf{CSP1_{ct}}] \\
=& P;\ Q
\end{array}
$$

□

A.6 Property of $\mathbf{CSP2_{ct}}$

Law 36. $\mathbf{CSP2_{ct}} \circ \mathbf{CSP2_{ct}} = \mathbf{CSP2_{ct}}$

Proof.

$$
\begin{aligned}
&\mathbf{CSP2_{ct}} \circ \mathbf{CSP2_{ct}}(P) && [\mathbf{CSP2_{ct}}]\\
=&P;\, J;\, J && [\text{property of } J]\\
=&P;\, J && [\mathbf{CSP2_{ct}}]\\
=&\mathbf{CSP2_{ct}}(P)
\end{aligned}
$$

□

Law 37. $\mathbf{CSP2_{ct}}(P \vee Q) = P \vee Q$ provided P and Q are $\mathbf{CSP2_{ct}}$

Proof.

$$
\begin{aligned}
&\mathbf{CSP2_{ct}}(P \vee Q) && [\mathbf{CSP2_{ct}}]\\
=&(P \vee Q);\, J && [\text{relational calculus}]\\
=&P;\, J \vee Q;\; J && [\mathbf{CSP2_{ct}}]\\
=&P;\, Q
\end{aligned}
$$

□

Law 38. $\mathbf{CSP2_{ct}}(P \lhd b \rhd Q) = P \lhd b \rhd Q$ provided P and Q are $\mathbf{CSP2_{ct}}$

Proof.

$$
\begin{aligned}
&\mathbf{CSP2_{ct}}(P \lhd b \rhd Q) && [\mathbf{CSP2_{ct}} \text{ and } \lhd\rhd]\\
=&((P \wedge b) \vee (Q \wedge \neg\, b));\, J && [\text{relational calculus}]\\
=&((P;\, J) \wedge b) \vee ((Q;\, J) \wedge \neg\, b) && [\mathbf{CSP2_{ct}} \text{ and } \lhd\rhd]\\
=&P \lhd b \rhd Q
\end{aligned}
$$

□

Law 39. $\mathbf{CSP2_{ct}}(P\,;\, Q) = P\,;\, Q$ provided Q is $\mathbf{CSP2_{ct}}$

Proof.

$$
\begin{aligned}
&\mathbf{CSP2_{ct}}(P\,;\, Q) && [\mathbf{CSP2_{ct}}]\\
=&P;\, Q;\, J && [\mathbf{CSP2_{ct}}]\\
=&P;\, \mathbf{CSP2_{ct}}(Q) && [Q \text{ is } \mathbf{CSP2_{ct}}]\\
=&P;\, Q
\end{aligned}
$$

□

Law 40. $P\,;\, J = P^f \vee (P^t \wedge ok')$

Proof.

$$
\begin{aligned}
&P;\, J && [J]\\
=&P;\, (ok \Rightarrow ok' \wedge wait' = wait \wedge tr' = tr \wedge ref' = ref \wedge state' = state) && [\text{relational calculus}]\\
=&(P;\, (\neg\, ok \wedge \mathbb{I}^{-ok})) \vee (P;\, (\mathbb{I}^{-ok} \wedge ok')) && [\text{relational calculus}]\\
=&(P;\, (\neg\, ok \wedge \mathbb{I}^{-ok})) \vee ((P;\, (\mathbb{I}^{-ok}) \wedge ok') && [\text{right one-point, twice}]\\
=&P^f \vee (P^t \wedge ok')
\end{aligned}
$$

□

B Proofs in transformation

B.1 Proofs of Property 3

L7. $L(Expands(tr_t, tr'_t)) = (tr \preccurlyeq tr' \wedge front(ref) \leq ref' \wedge \#diff(tr', tr) = \#(ref' - front(ref))$

Proof.

$$
\begin{aligned}
& L(Expands(tr_t, tr'_t)) && \text{[def of } L\text{]}\\
&= \exists\, tr_t, tr'_t \bullet Expands(tr_t, tr'_t) \wedge \begin{pmatrix} tr = proj\_fst(tr_t) \wedge tr' = proj\_fst(tr'_t) \wedge \\ ref = proj\_snd(tr_t) \wedge ref' = proj\_snd(tr'_t) \end{pmatrix} && [Expands(2.2)]\\
&= \exists\, tr_t, tr'_t \bullet \begin{pmatrix} front(tr_t) \leq tr'_t \wedge \\ fst(last(tr_t)) \leq fst(tr'_t(\#tr_t)) \end{pmatrix} \wedge \begin{pmatrix} tr = proj\_fst(tr_t) \wedge tr' = proj\_fst(tr'_t) \wedge \\ ref = proj\_snd(tr_t) \wedge ref' = proj\_snd(tr'_t) \end{pmatrix} && \text{[Property 2-L1]}\\
&= \exists\, tr_t, tr'_t \bullet \begin{pmatrix} front(proj\_fst(tr_t)) \leq proj\_fst(tr'_t) \wedge \\ front(proj\_snd(tr_t)) \leq proj\_snd(tr'_t) \wedge \\ fst(last(tr_t)) \leq fst(tr'_t(\#tr_t)) \end{pmatrix} \wedge \begin{pmatrix} tr = proj\_fst(tr_t) \wedge tr' = proj\_fst(tr'_t) \wedge \\ ref = proj\_snd(tr_t) \wedge ref' = proj\_snd(tr'_t) \end{pmatrix} && \text{[Property 2-L2,L3,L4]}\\
&= \exists\, tr_t, tr'_t \bullet \begin{pmatrix} front(proj\_fst(tr_t)) \leq proj\_fst(tr'_t) \wedge \\ front(proj\_snd(tr_t)) \leq proj\_snd(tr'_t) \wedge \\ last(proj\_fst(tr_t)) \leq proj\_fst(tr'_t)(\#proj\_fst(tr_t)) \end{pmatrix} \wedge \begin{pmatrix} tr = proj\_fst(tr_t) \wedge \\ tr' = proj\_fst(tr'_t) \wedge \\ ref = proj\_snd(tr_t) \wedge \\ ref' = proj\_snd(tr'_t) \end{pmatrix} && \text{[Property 2-L4]}\\
&= \exists\, tr_t, tr'_t \bullet \begin{pmatrix} front(proj\_fst(tr_t)) \leq proj\_fst(tr'_t) \wedge \\ front(proj\_snd(tr_t)) \leq proj\_snd(tr'_t) \wedge \\ last(proj\_fst(tr_t)) \leq proj\_fst(tr'_t)(\#proj\_fst(tr_t)) \end{pmatrix} \wedge \begin{pmatrix} tr = proj\_fst(tr_t) \wedge \\ tr' = proj\_fst(tr'_t) \wedge \\ ref = proj\_snd(tr_t) \wedge \\ ref' = proj\_snd(tr'_t) \end{pmatrix}\\
&\quad \wedge \begin{pmatrix} \#proj\_fst(tr_t) = \#proj\_snd(tr_t) \wedge \\ \#proj\_fst(tr'_t) = \#proj\_snd(tr'_t) \end{pmatrix} && \text{[substitution]}\\
&= \exists\, tr_t, tr'_t \bullet (front(tr) \leq tr' \wedge last(tr) \leq tr'(\#tr) \wedge front(ref) \leq ref' \wedge \#tr = \#ref \wedge \#tr' = \#ref')\\
&\quad \wedge \begin{pmatrix} tr = proj\_fst(tr_t) \wedge tr' = proj\_fst(tr'_t) \wedge \\ ref = proj\_snd(tr_t) \wedge ref' = proj\_snd(tr'_t) \end{pmatrix} && \text{[predicate calculus]}\\
&= (front(tr) \leq tr' \wedge last(tr) \leq tr'(\#tr) \wedge front(ref) \leq ref' \wedge \#tr = \#ref \wedge \#tr' = \#ref')\\
&\quad \wedge \exists\, tr_t, tr'_t \bullet \left(\; tr = proj\_fst(tr_t) \wedge tr' = proj\_fst(tr'_t) \wedge ref = proj\_snd(tr_t) \wedge ref' = proj\_snd(tr'_t) \; \right)\\
&(1)\text{[propositional calculus]}\\
&\Rightarrow (front(tr) \leq tr' \wedge last(tr) \leq tr'(\#tr) \wedge front(ref) \leq ref' \wedge \#tr = \#ref \wedge \#tr' = \#ref') && \text{[def of } \preccurlyeq\text{]}\\
&= tr \preccurlyeq tr'_t \wedge front(ref) \leq ref' \wedge \#tr = \#ref \wedge \#tr' = \#ref'\\
&(2)\text{[Let } tr_t = tr \circledast ref \text{ and } tr'_t = tr' \circledast ref'\text{]}\\
&\Leftarrow front(tr) \leq tr'_t \wedge last(tr) \leq tr'(\#tr) \wedge front(ref) \leq ref' \wedge \#tr = \#ref \wedge \#tr' = \#ref'\\
&\quad \wedge \begin{pmatrix} tr = proj\_fst(tr \circledast ref) \wedge tr' = proj\_fst(tr' \circledast ref') \wedge \\ ref = proj\_snd(tr \circledast ref) \wedge ref' = proj\_snd(tr' \circledast ref') \end{pmatrix} && \text{[Law 41, 42]}\\
&= front(tr) \leq tr'_t \wedge last(tr) \leq tr'(\#tr) \wedge front(ref) \leq ref' \wedge \#tr = \#ref \wedge \#tr' = \#ref' && \text{[def of } \preccurlyeq\text{]}\\
&= tr \preccurlyeq tr'_t \wedge front(ref) \leq ref' \wedge \#tr = \#ref \wedge \#tr' = \#ref'
\end{aligned}
$$

□

L8. $L(tr'_t = tr_t) = (tr' = tr \wedge ref' = ref \wedge \#tr = \#ref \wedge \#tr' = \#ref')$

Proof.

$$
\begin{aligned}
& L(tr_t' = tr_t) && \text{[def of } L\text{]}\\
=\ & \exists\, tr_t, tr_t' \bullet tr_t = tr_t' \wedge \begin{pmatrix} tr = proj\_fst(tr_t) \wedge tr' = proj\_fst(tr_t') \wedge \\ ref = proj\_snd(tr_t) \wedge ref' = proj\_snd(tr_t') \end{pmatrix} && \text{[Property 2-L5]}\\
=\ & \exists\, tr_t, tr_t' \bullet \begin{pmatrix} proj\_fst(tr_t) = proj\_fst(tr_t') \wedge \\ proj\_snd(tr_t) = proj\_snd(tr_t') \end{pmatrix} \wedge \begin{pmatrix} tr = proj\_fst(tr_t) \wedge tr' = proj\_fst(tr_t') \wedge \\ ref = proj\_snd(tr_t) \wedge ref' = proj\_snd(tr_t') \end{pmatrix} && \text{[Property 2-L4]}\\
=\ & \exists\, tr_t, tr_t' \bullet \begin{pmatrix} proj\_fst(tr_t) = proj\_fst(tr_t') \wedge \\ proj\_snd(tr_t) = proj\_snd(tr_t') \end{pmatrix} \wedge \begin{pmatrix} tr = proj\_fst(tr_t) \wedge tr' = proj\_fst(tr_t') \wedge \\ ref = proj\_snd(tr_t) \wedge ref' = proj\_snd(tr_t') \end{pmatrix} \\
& \wedge \left(\#proj\_fst(tr_t) = \#proj\_snd(tr_t) \wedge \#proj\_fst(tr_t') = \#proj\_snd(tr_t') \right) && \text{[substitution and predicate calculus]}\\
=\ & tr' = tr \wedge ref' = ref \wedge \#tr = \#ref \wedge \#tr' = \#ref' \wedge \\
& \exists\, tr_t, tr_t' \bullet \left(tr = proj\_fst(tr_t) \wedge tr' = proj\_fst(tr_t') \wedge ref = proj\_snd(tr_t) \wedge ref' = proj\_snd(tr_t') \right)\\
(1)\ & [\text{[propositional calculus]}\\
\Rightarrow\ & tr' = tr \wedge ref' = ref \wedge \#tr = \#ref \wedge \#tr' = \#ref'\\
(2)\ & [\text{Let } tr_t = tr \circledast ref \text{ and } tr_t' = tr' \circledast ref']\\
\Leftarrow\ & tr' = tr \wedge ref' = ref \wedge \#tr = \#ref \wedge \#tr' = \#ref' \wedge \\
& \wedge \begin{pmatrix} tr = proj\_fst(tr \circledast ref) \wedge tr' = proj\_fst(tr' \circledast ref') \wedge \\ ref = proj\_snd(tr \circledast ref) \wedge ref' = proj\_snd(tr' \circledast ref') \end{pmatrix} && \text{[Law 41, 42]}\\
=\ & tr' = tr \wedge ref' = ref \wedge \#tr = \#ref \wedge \#tr' = \#ref' =
\end{aligned}
$$

□

L9. $L(\mathit{II}_t) = \mathit{II}_{ct}$

Proof.

$$
\begin{aligned}
& L(\mathit{II}_t) && [\ \mathit{II}_t(2.7)]\\
=\ & L((\neg\ ok \wedge Expands(tr_t, tr_t')) \vee (ok' \wedge wait' = wait \wedge state' = state \wedge tr_t = tr_t')) && [L \text{ and Property 3-L2,L4}]\\
=\ & (\neg\ ok \wedge L(Expands(tr_t, tr_t'))) \vee (ok' \wedge wait' = wait \wedge state' = state \wedge L(tr_t = tr_t')) && \text{[Property 3-L7,L8]}\\
=\ & (\neg\ ok \wedge tr \preccurlyeq tr_t' \wedge front(ref) \leq ref' \wedge \#tr = \#ref \wedge \#tr' = \#ref') \vee && [\mathit{II}_{ct}]\\
& (ok' \wedge wait' = wait \wedge state' = state \wedge tr' = tr \wedge ref' = ref \wedge \#tr = \#ref \wedge \#tr' = \#ref')\\
=\ & \mathit{II}_{ct}
\end{aligned}
$$

□

L10. $L(trace' = t) = (\frown/tr' - \frown/tr = t)$

Proof.

$$
\begin{array}{ll}
& L(trace' = t) \qquad [2.1] \\
= & L(Flat(tr_t') - Flat(tr_t) = t) \\
= & \exists\, tr_t, tr_t' \bullet \big(tr = proj\_fst(tr_t) \wedge tr' = proj\_fst(tr_t') \wedge ref = proj\_snd(tr_t) \wedge ref' = proj\_snd(tr_t') \big) \\
& \wedge\, Flat(tr_t') - Flat(tr_t) = t \qquad [\text{Property 2-L6}] \\
= & \exists\, tr_t, tr_t' \bullet \big(tr = proj\_fst(tr_t) \wedge tr' = proj\_fst(tr_t') \wedge ref = proj\_snd(tr_t) \wedge ref' = proj\_snd(tr_t') \big) \\
& \wedge\, {}^\frown\!/(proj\_fst(tr_t')) - {}^\frown\!/(proj\_fst(tr_t)) = t \qquad [\text{substitution}] \\
= & \exists\, tr_t, tr_t' \bullet \begin{pmatrix} tr = proj\_fst(tr_t) \wedge tr' = proj\_fst(tr_t') \wedge \\ ref = proj\_snd(tr_t) \wedge ref' = proj\_snd(tr_t') \end{pmatrix} \wedge {}^\frown\!/tr' - {}^\frown\!/tr = t
\end{array}
$$

(1)[propositional calculus]

$$
\begin{array}{l}
\Rightarrow {}^\frown\!/tr' - {}^\frown\!/tr = t
\end{array}
$$

(2)[Let $tr_t = \langle tr, ref \rangle$ and $tr_t' = \langle tr', ref' \rangle$]

$$
\begin{array}{l}
\Leftarrow {}^\frown\!/tr' - {}^\frown\!/tr = t \wedge \exists\, tr_t, tr_t' \bullet \begin{pmatrix} tr = proj\_fst(\langle (tr, ref) \rangle) \wedge tr' = proj\_fst(\langle (tr', ref') \rangle) \wedge \\ ref = proj\_snd(\langle (tr, ref) \rangle) \wedge ref' = proj\_snd(\langle (tr', ref') \rangle) \end{pmatrix} \\
\qquad\qquad [\text{predicate calculus}] \\
= {}^\frown\!/tr' - {}^\frown\!/tr = t
\end{array}
$$

□

L11. $L(wait\_com(c)_t) = wait\_com(c)_{ct}$

Proof.

$$
\begin{array}{ll}
& L(wait\_com(c)_t) \qquad [wait\_com(c)(4.55) \text{ and } possible(2.25)] \\
= & L(wait' \wedge \forall\, i : \#tr_t..\#tr_t' \bullet c \notin snd(tr_t'(i)) \wedge trace' = \langle\rangle) \qquad [\text{Property 3-L4}] \\
= & wait' \wedge L(\forall\, i : \#tr_t..\#tr_t' \bullet c \notin snd(tr_t'(i)) \wedge trace' = \langle\rangle) \qquad [\text{def of } L] \\
= & wait' \wedge \exists\, tr_t, tr_t' \bullet \begin{pmatrix} tr = proj\_fst(tr_t) \wedge tr' = proj\_fst(tr_t') \wedge \\ ref = proj\_snd(tr_t) \wedge ref' = proj\_snd(tr_t') \end{pmatrix} \wedge \\
& \forall\, i : \#tr_t..\#tr_t' \bullet c \notin snd(tr_t'(i)) \wedge trace' = \langle\rangle \qquad [\text{Property 3-L10}] \\
= & wait' \wedge \exists\, tr_t, tr_t' \bullet \begin{pmatrix} tr = proj\_fst(tr_t) \wedge tr' = proj\_fst(tr_t') \wedge \\ ref = proj\_snd(tr_t) \wedge ref' = proj\_snd(tr_t') \end{pmatrix} \wedge \\
& \forall\, i : \#tr_t..\#tr_t' \bullet c \notin snd(tr_t'(i)) \wedge ({}^\frown\!/tr' - {}^\frown\!/tr = \langle\rangle) \qquad [\text{Property 2-L3,L7}] \\
= & wait' \wedge \exists\, tr_t, tr_t' \bullet \begin{pmatrix} tr = proj\_fst(tr_t) \wedge tr' = proj\_fst(tr_t') \wedge \\ ref = proj\_snd(tr_t) \wedge ref' = proj\_snd(tr_t') \end{pmatrix} \wedge \qquad [\text{substitution}] \\
& \forall\, i : \#proj\_fst(tr_t)..\#proj\_fst(tr_t') \bullet c \notin proj\_snd(tr_t')(i) \wedge ({}^\frown\!/tr' - {}^\frown\!/tr = \langle\rangle) \\
= & wait' \wedge \exists\, tr_t, tr_t' \bullet \begin{pmatrix} tr = proj\_fst(tr_t) \wedge tr' = proj\_fst(tr_t') \wedge \\ ref = proj\_snd(tr_t) \wedge ref' = proj\_snd(tr_t') \end{pmatrix} \wedge \\
& \forall\, i : \#tr..\#tr' \bullet c \notin ref'(i) \wedge ({}^\frown\!/tr' - {}^\frown\!/tr = \langle\rangle)
\end{array}
$$

(1)[propositional calculus]

$$
\begin{array}{l}
\Rightarrow wait' \wedge \forall\, i : \#tr..\#tr' \bullet c \notin ref'(i) \wedge ({}^\frown\!/tr' - {}^\frown\!/tr = \langle\rangle) \\
= wait\_com(c)_{ct}
\end{array}
$$

(2)[Let $tr_t = tr \circledast ref$ and $tr_t' = tr' \circledast ref'$]

$$
\begin{array}{ll}
\Leftarrow & wait' \wedge \forall\, i : \#tr..\#tr' \bullet c \notin ref'(i) \wedge ({}^\frown\!/tr' - {}^\frown\!/tr = \langle\rangle) \wedge \\
& \wedge \begin{pmatrix} tr = proj\_fst(tr \circledast ref) \wedge tr' = proj\_fst(tr' \circledast ref') \wedge \\ ref = proj\_snd(tr \circledast ref) \wedge ref' = proj\_snd(tr' \circledast ref') \end{pmatrix} \qquad [\text{Law 41, 42}] \\
= & wait' \wedge \forall\, i : \#tr..\#tr' \bullet c \notin ref'(i) \wedge ({}^\frown\!/tr' - {}^\frown\!/tr = \langle\rangle) \qquad [wait\_com(c)_{ct}] \\
= & wait\_com(c)_{ct}
\end{array}
$$

□

L12. $L(term\_com(c.e)_t) = term\_com(c.e)_{ct}$

Proof.

$$
\begin{aligned}
& L(term\_com(c.e)_t) && [term\_com(c)(4.57)]\\
=& L(\neg\ wait' \wedge trace' = \langle c.e\rangle \wedge \#tr'_t = \#tr_t) && [\text{Property 3-L4}]\\
=& \neg\ wait' \wedge L(trace' = \langle c.e\rangle \wedge \#tr'_t = \#tr_t) && [L]\\
=& \neg\ wait' \wedge \exists\, tr_t, tr'_t \bullet \begin{pmatrix} tr = proj\_fst(tr_t) \wedge tr' = proj\_fst(tr'_t) \wedge \\ ref = proj\_snd(tr_t) \wedge ref' = proj\_snd(tr'_t) \end{pmatrix} \wedge trace' = \langle c.e\rangle \wedge \#tr'_t = \#tr_t\\
& && [\text{Property 2-L4 and Property 3-L10}]\\
=& \neg\ wait' \wedge \exists\, tr_t, tr'_t \bullet \begin{pmatrix} tr = proj\_fst(tr_t) \wedge tr' = proj\_fst(tr'_t) \wedge \\ ref = proj\_snd(tr_t) \wedge ref' = proj\_snd(tr'_t) \end{pmatrix} \wedge {}^\frown/tr' - {}^\frown/tr = \langle c.e\rangle \wedge\\
& \#proj\_fst(tr'_t) = \#proj\_fst(tr_t) && [\text{substitution and preidcate calculus}]\\
=& \neg\ wait' \wedge {}^\frown/tr' - {}^\frown/tr = \langle c.e\rangle \wedge \#tr' = \#tr \wedge\\
& \exists\, tr_t, tr'_t \bullet \big(\, tr = proj\_fst(tr_t) \wedge tr' = proj\_fst(tr'_t) \wedge ref = proj\_snd(tr_t) \wedge ref' = proj\_snd(tr'_t) \,\big)\\
& (1)[\text{propositional calculus}]\\
\Rightarrow& \neg\ wait' \wedge {}^\frown/tr' - {}^\frown/tr = \langle c.e\rangle \wedge \#tr' = \#tr && [term\_com(c.e)_{ct}]\\
=& term\_com(c.e)_{ct}\\
& (2)[\text{Let } tr_t = tr \circledast ref \text{ and } tr'_t = tr' \circledast ref']\\
\Leftarrow& \neg\ wait' \wedge {}^\frown/tr' - {}^\frown/tr = \langle c.e\rangle \wedge \#tr' = \#tr \wedge\\
& \wedge \begin{pmatrix} tr = proj\_fst(tr \circledast ref) \wedge tr' = proj\_fst(tr' \circledast ref') \wedge \\ ref = proj\_snd(tr \circledast ref) \wedge ref' = proj\_snd(tr' \circledast ref') \end{pmatrix} && [\text{Law 41, 42}]\\
=& \neg\ wait' \wedge {}^\frown/tr' - {}^\frown/tr = \langle c.e\rangle \wedge \#tr' = \#tr && [\text{Property 5-L14}]\\
=& \neg\ wait \wedge diff(tr', tr) = \langle\langle c.e\rangle\rangle && [term\_com(c.e)_{ct}]\\
=& term\_com(c)_{ct}
\end{aligned}
$$

□

L13. $L(terminating\_com(c.e)_t) = terminating\_com(c.e)_{ct}$

Proof.

$$
\begin{aligned}
& L(terminating\_com(c.e)_t) && [terminating\_com(c.e)_t(4.59)]\\
=& L(wait\_com_t(c)\,;\, term\_com_t(c.e) \vee term\_com_t(c.e)) && [\text{Property 3-L2,L6}]\\
=& L(wait\_com_t(c))\,;\, L(term\_com_t(c.e)) \vee L(term\_com_t(c.e)) && [\text{Property 3-L11,L12}]\\
=& wait\_com_t(c)\,;\, term\_com_t(c.e) \vee term\_com_t(c.e) && [terminating\_com_{ct}]\\
=& terminating\_com(c.e)_{ct}
\end{aligned}
$$

□

L14.

$$
\begin{aligned}
& L(dif(tr'_t, tr_t) \in TSync(dif(0.tr'_t, tr_t), dif(1.tr'_t, tr_t), CS)) =\\
& \begin{pmatrix} diff(tr', tr) \in SSync(diff(0.tr', tr), diff(1.tr', tr), CS) \wedge \\ ref' - front(ref) = MRef(0.ref' - front(ref), 1.ref' - front(ref), CS) \end{pmatrix}
\end{aligned}
$$

Proof. The proof is based on the induction rules on timed traces and will not be given here. □

L15. $L(M_t) = M_{ct}$

Proof.

$$
\begin{aligned}
& L(M_t) && \text{[2.34]}\\
&= L\left(\begin{array}{c} ok' = (0.ok \wedge 1.ok) \wedge wait' = (0.wait \vee 1.wait) \wedge \\ dif(tr'_t, tr_t) \in TSync(dif(0.tr_t, tr_t), dif(1.tr_t, tr_t), CS) \\ \wedge\ state' = (0.state - s_2) \oplus (1.state - s_1) \end{array}\right) && \text{[Property 3-L4]}\\
&= \left(\begin{array}{c} ok' = (0.ok \wedge 1.ok) \wedge wait' = (0.wait \vee 1.wait) \wedge \\ L(dif(tr'_t, tr_t) \in TSync(dif(0.tr_t, tr_t), dif(1.tr_t, tr_t), CS)) \\ \wedge\ state' = (0.state - s_2) \oplus (1.state - s_1) \end{array}\right) && \text{[Property 3-L14]}\\
&= \left(\begin{array}{c} ok' = (0.ok \wedge 1.ok) \wedge wait' = (0.wait \vee 1.wait) \wedge \\ diff(tr', tr) \in SSync(diff(0.tr', tr), diff(1.tr', tr), CS) \wedge \\ ref' - front(ref) = MRef(0.ref' - front(ref), 1.ref' - front(ref), CS) \\ \wedge\ state' = (0.state - s_2) \oplus (1.state - s_1) \end{array}\right) && \text{[def of } M_{ct}\text{]}\\
&= M_{ct}
\end{aligned}
$$

□

L16.

$$
L(dif(tr'_t, tr_t) = dif(s_t, tr_t) \downarrow_t (Event - CS)) = \left(\begin{array}{c} diff(tr', tr) = diff(s, tr) \downarrow_t (\Sigma - CS) \wedge \\ r - front(ref) = ((ref' - front(ref)) \cup_t CS \end{array}\right)
$$

Proof. The proof is based on the induction rules on timed traces and will not be given here. □

L17.

$$
L(DifDetected(P, Q)_t) \Rightarrow DifDetected(L(P), L(Q))_{ct}
$$

Proof.

$$
\begin{aligned}
& L(DifDetected(P, Q))\\
&= L\left(\neg\ ok' \vee \left(ok \wedge \neg\ wait \wedge \left(\left(\begin{array}{l} P \wedge Q \wedge ok' \wedge \\ wait' \wedge trace' = \langle\rangle \end{array}\right) \vee Skip_t\right)\right); \left(\begin{array}{l} (ok' \wedge \neg\ wait' \wedge tr'_t = tr_t) \vee \\ (ok' \wedge fst(head(dif(tr'_t, tr_t))) \neq \langle\rangle) \end{array}\right)\right)\\
& \qquad \text{[Property 3-L2,L6]}\\
&= \neg\ ok' \vee L\left(ok \wedge \neg\ wait \wedge \left(\left(\begin{array}{l} P \wedge Q \wedge ok' \wedge \\ wait' \wedge trace' = \langle\rangle \end{array}\right) \vee Skip_t\right)\right); L\left(\begin{array}{l} (ok' \wedge \neg\ wait' \wedge tr'_t = tr_t) \vee \\ (ok' \wedge fst(head(dif(tr'_t, tr_t))) \neq \langle\rangle) \end{array}\right)\\
& \qquad \text{[Property 3-L2,L3]}\\
&\Rightarrow \neg\ ok' \vee \left(ok \wedge \neg\ wait \wedge \left(\left(\begin{array}{l} L(P) \wedge L(Q) \wedge ok' \wedge wait' \\ \wedge\ L(trace' = \langle\rangle) \end{array}\right) \vee L(Skip_t)\right)\right); L\left(\begin{array}{l} (ok' \wedge \neg\ wait' \wedge L(tr'_t = tr_t)) \vee \\ (ok' \wedge L(fst(head(dif(tr'_t, tr_t))) \neq \langle\rangle)) \end{array}\right)\\
& \qquad \text{[Property 3-L8,L10 and Law 45]}\\
&= \neg\ ok' \vee \left(ok \wedge \neg\ wait \wedge \left(\left(\begin{array}{l} L(P) \wedge L(Q) \wedge ok' \wedge wait' \\ \wedge\ {}^\frown/tr' = {}^\frown/tr \end{array}\right) \vee Skip_{ct}\right)\right);\\
& \qquad \left(\begin{array}{l} (ok' \wedge \neg\ wait' \wedge tr' = tr \wedge ref' = ref \wedge \#tr = \#ref \wedge \#tr' = \#ref') \vee \\ (ok' \wedge L(fst(head(dif(tr'_t, tr_t))) \neq \langle\rangle)) \end{array}\right) \quad \text{[Property of } L \text{ and } dif\text{]}\\
&= \neg\ ok' \vee \left(ok \wedge \neg\ wait \wedge \left(\left(\begin{array}{l} L(P) \wedge L(Q) \wedge ok' \wedge wait' \\ \wedge\ {}^\frown/tr' = {}^\frown/tr \end{array}\right) \vee Skip_{ct}\right)\right); \qquad \text{[}DifDetected_{ct}\text{]}\\
& \qquad ((ok' \wedge \neg\ wait' \wedge tr' = tr \wedge ref' = ref \wedge \#tr = \#ref \wedge \#tr' = \#ref') \vee (ok' \wedge head(diff(tr', tr)) \neq \langle\rangle)))\\
&= DifDetected(L(P), L(Q))_{ct}
\end{aligned}
$$

□

B.2 Proofs of Property 4

L1. $L(\mathbf{R1_t}(X)) = \mathbf{R1_{ct}}(L(X))$

Proof.

$$
\begin{aligned}
& L(\mathbf{R1_t}(X)) && [\mathbf{R1_t}(2.3)] \\
=& L(X \wedge \mathit{Expands}(tr_t, tr_t')) && [\mathit{Expands}(2.2)] \\
=& L\left(X \wedge \left(\mathit{front}(tr_t) \leq tr_t' \wedge \mathit{fst}(\mathit{last}(tr_t)) \leq \mathit{fst}(tr_t'(\#tr_t))\right)\right) && [\text{def of } L] \\
=& \exists\, tr_t, tr_t' \bullet \left(\begin{array}{c} tr = \mathit{proj\_fst}(tr_t) \wedge tr' = \mathit{proj\_fst}(tr_t') \wedge \\ ref = \mathit{proj\_snd}(tr_t) \wedge ref' = \mathit{proj\_snd}(tr_t') \end{array}\right) \wedge X && [\text{Property 3-L7}] \\
& \wedge \left(\mathit{front}(tr_t) \leq tr_t' \wedge \mathit{fst}(\mathit{last}(tr_t)) \leq \mathit{fst}(tr_t'(\#tr_t))\right) \\
=& \exists\, tr_t, tr_t' \bullet \left(\begin{array}{c} tr = \mathit{proj\_fst}(tr_t) \wedge tr' = \mathit{proj\_fst}(tr_t') \wedge \\ ref = \mathit{proj\_snd}(tr_t) \wedge ref' = \mathit{proj\_snd}(tr_t') \end{array}\right) \wedge X && [\text{def of } L] \\
& \wedge tr \preccurlyeq tr_t' \wedge \mathit{front}(ref) \leq ref' \wedge \#tr = \#ref \wedge \#tr' = \#ref' \\
=& L(X) \wedge tr \preccurlyeq tr_t' \wedge \mathit{front}(ref) \leq ref' \wedge \#tr = \#ref \wedge \#tr' = \#ref' && [\mathbf{R1_t}] \\
=& \mathbf{R1_t}(L(X))
\end{aligned}
$$

□

L2. $L(\mathbf{R2_t}(X)) \Rightarrow \mathbf{R2_{ct}}(L(X))$

Proof.

$$
\begin{aligned}
& L(\mathbf{R2_t}(X)) && [\mathbf{R2_t}(2.4)]\\
=& L(\exists\, r \bullet X[\langle(\langle\rangle, r)\rangle, dif(tr'_t, tr_t)/tr_t, tr'_t]) && [\text{substitution}]\\
=& L(\exists\, tr_t, tr'_t, r \bullet X \wedge tr_t = \langle(\langle\rangle, r)\rangle \wedge tr'_t = dif(tr'_t, tr_t)) && [\text{def of } L]\\
=& \exists\, tr_t, tr'_t, r \bullet f(tr_t, tr'_t, tr, tr', ref, ref') \wedge X \wedge tr_t = \langle(\langle\rangle, r)\rangle \wedge tr'_t = dif(tr'_t, tr_t)) && [dif(2.5)]\\
=& \exists\, tr_t, tr'_t, r \bullet f(tr_t, tr'_t, tr, tr', ref, ref') \wedge X \wedge tr_t = \langle(\langle\rangle, r)\rangle \wedge && [\text{Property 2-L5}]\\
& tr'_t = \langle(fst(head(tr'_t - front(tr_t))) - fst(last(tr_t)), snd(head(tr'_t - front(tr_t))))\rangle \frown tail(tr'_t - front(tr_t))\\
=& \exists\, tr_t, tr'_t, r \bullet f(tr_t, tr'_t, tr, tr', ref, ref') \wedge X \wedge proj\_fst(tr_t) = \langle\langle\rangle\rangle \wedge proj\_snd(tr_t) = r \wedge\\
& proj\_fst(tr'_t) = \langle fst(head(tr'_t - front(tr_t))) - fst(last(tr_t))\rangle \frown proj\_fst(tail(tr'_t - front(tr_t))) \wedge\\
& proj\_snd(tr'_t) = \langle snd(head(tr'_t - front(tr_t)))\rangle \frown proj\_snd(tail(tr'_t - front(tr_t)))\\
& && [\text{def of } f \text{ and Property 2-L9,L10,L11,L12}]\\
=& \exists\, tr_t, tr'_t, r \bullet \begin{pmatrix} tr = proj\_fst(tr_t) \wedge tr' = proj\_fst(tr'_t) \wedge \\ ref = proj\_snd(tr_t) \wedge ref' = proj\_snd(tr'_t) \end{pmatrix} \wedge && [\text{substitution}]\\
& X \wedge proj\_fst(tr_t) = \langle\langle\rangle\rangle \wedge proj\_snd(tr_t) = \langle r\rangle \wedge\\
& proj\_fst(tr'_t) = \langle fst(head(tr'_t - front(tr_t))) - fst(last(tr_t))\rangle \frown tail(proj\_fst(tr'_t) - front(proj\_fst(tr_t)))\\
& \wedge proj\_snd(tr'_t) = \langle snd(head(tr'_t - front(tr_t)))\rangle \frown tail(proj\_snd(tr'_t) - front(proj\_snd(tr_t)))\\
=& \exists\, tr_t, tr'_t, r \bullet \begin{pmatrix} tr = proj\_fst(tr_t) \wedge tr' = proj\_fst(tr'_t) \wedge \\ ref = proj\_snd(tr_t) \wedge ref' = proj\_snd(tr'_t) \end{pmatrix} \wedge X \wedge tr = \langle\langle\rangle\rangle \wedge ref = r \wedge\\
& tr' = \langle fst(head(tr'_t - front(tr_t))) - fst(last(tr_t))\rangle \frown tail(tr' - front(tr))\\
& \wedge ref' = \langle snd(head(tr'_t - front(tr_t)))\rangle \frown tail(ref' - front(ref)) && [\text{Property A.1-L12}]\\
=& \exists\, tr_t, tr'_t, r \bullet \begin{pmatrix} tr = proj\_fst(tr_t) \wedge tr' = proj\_fst(tr'_t) \wedge \\ ref = proj\_snd(tr_t) \wedge ref' = proj\_snd(tr'_t) \end{pmatrix} \wedge X \wedge tr = \langle\langle\rangle\rangle \wedge ref = \langle r\rangle \wedge\\
& tr' = \langle fst(tr'_t(\#tr_t)) - fst(last(tr_t))\rangle \frown tail(tr' - front(tr)) \wedge\\
& ref' = \langle snd(tr'_t(\#tr_t))\rangle \frown tail(ref' - front(ref)) && [\text{Property 2-L2,L7,L13}]\\
=& \exists\, tr_t, tr'_t, r \bullet \begin{pmatrix} tr = proj\_fst(tr_t) \wedge tr' = proj\_fst(tr'_t) \wedge \\ ref = proj\_snd(tr_t) \wedge ref' = proj\_snd(tr'_t) \end{pmatrix} \wedge X \wedge tr = \langle\langle\rangle\rangle \wedge ref = r \wedge\\
& tr' = \langle proj\_fst(tr'_t(\#tr_t)) - last(proj\_fst(tr_t))\rangle \frown tail(tr' - front(tr)) \wedge\\
& ref' = \langle proj\_snd(tr'_t(\#tr_t))\rangle \frown tail(ref' - front(ref)) && [\text{substitution and Property 2-L4}]\\
=& \exists\, tr_t, tr'_t, r \bullet \begin{pmatrix} tr = proj\_fst(tr_t) \wedge tr' = proj\_fst(tr'_t) \wedge \\ ref = proj\_snd(tr_t) \wedge ref' = proj\_snd(tr'_t) \end{pmatrix} \wedge X \wedge tr = \langle\langle\rangle\rangle \wedge ref = r \wedge\\
& tr' = \langle tr'(\#tr) - last(tr)\rangle \frown tail(tr' - front(tr))\\
& \wedge ref' = \langle ref'(\#ref)\rangle \frown tail(ref' - front(ref)) && [\text{def of } diff \text{ and Property A.1-L13}]\\
=& \exists\, tr_t, tr'_t, r \bullet \begin{pmatrix} tr = proj\_fst(tr_t) \wedge tr' = proj\_fst(tr'_t) \wedge \\ ref = proj\_snd(tr_t) \wedge ref' = proj\_snd(tr'_t) \end{pmatrix} \wedge X \wedge tr = \langle\langle\rangle\rangle \wedge ref = r \wedge\\
& tr' = diff(tr', tr) \wedge ref' = ref' - front(ref) && [L]\\
=& \exists\, r \bullet L(X) \wedge tr = \langle\rangle \wedge ref = r \wedge tr' = diff(tr', tr) \wedge ref' = ref' - front(ref)\\
\Rightarrow& L(X) \wedge tr = \langle\langle\rangle\rangle \wedge tr' = diff(tr', tr) && [\mathbf{R2_{ct}}]\\
=& \mathbf{R2_{ct}}(L(X))
\end{aligned}
$$

□

L3. $L(\mathbf{R3_t}(X)) = \mathbf{R3_{ct}}(L(X))$

Proof.

$$
\begin{array}{ll}
& L(\mathbf{R3_t}(X)) & [\mathbf{R3_t}(2.6)] \\
= & L(\mathit{II}_t \lhd \mathit{wait} \rhd X) & [\text{Property 3-L5}] \\
= & L(\mathit{II}_t) \lhd \mathit{wait} \rhd L(X) & [\text{Property 3-L9}] \\
= & \mathit{II}_{ct} \lhd \mathit{wait} \rhd L(X) & [\mathbf{R3_{ct}}] \\
= & \mathbf{R3_{ct}}(L(X))
\end{array}
$$

□

L4. $L(\mathbf{CSP1_t}(X)) = \mathbf{CSP1_{ct}}(L(X))$

Proof.

$$
\begin{array}{ll}
& L(\mathbf{CSP1_t}(X)) & [\mathbf{CSP1_t}(2.8)] \\
= & L(X \vee (\neg\ ok \wedge \mathit{Expands}(tr_t, tr_t'))) & [\text{Property 3-L2}] \\
= & L(X) \vee L(\neg\ ok \wedge \mathit{Expands}(tr_t, tr_t')) & [\text{Property 3-L4}] \\
= & L(X) \vee (\neg\ ok \wedge L(\mathit{Expands}(tr_t, tr_t'))) & [\text{Property 3-L7}] \\
= & L(X) \vee (\neg\ ok \wedge RT) & [\mathbf{CSP1_{ct}}] \\
= & \mathbf{CSP1_t}(L(X))
\end{array}
$$

□

L5. $L(\mathbf{CSP2_t}(X)) = \mathbf{CSP2_{ct}}(L(X))$

Proof.

$$
\begin{array}{ll}
& L(\mathit{CSP2}_s(X)) & [\mathbf{CSP2_t}(2.9)] \\
= & L(X\ ;\ J_t) & [\text{Property 3-L6}] \\
= & L(X)\ ;\ L(J_t) & [J_t(2.13)] \\
= & L(X)\ ;\ L\left(\begin{array}{c}(ok \Rightarrow ok') \wedge wait' = wait \wedge state' = state \wedge \\ front(tr_t) = front(tr_t') \wedge fst(last(tr_t)) = fst(last(tr_t'))\end{array}\right) & [\text{Property 3-L4}] \\
= & L(X)\ ;\ \left(\begin{array}{c}(ok \Rightarrow ok') \wedge wait' = wait \wedge state' = state \wedge \\ L(front(tr_t) = front(tr_t') \wedge fst(last(tr_t)) = fst(last(tr_t')))\end{array}\right) & [L \text{ and Property 2-L2,L8}] \\
= & L(X)\ ;\ \left(\begin{array}{c}(ok \Rightarrow ok') \wedge wait' = wait \wedge state' = state \wedge \\ \left(\begin{array}{c}\exists\, tr_t, tr_t' \bullet f(tr_t, tr_t', tr, tr', ref, ref') \wedge front(proj\_fst(tr_t)) = front(proj\_fst(tr_t')) \wedge \\ front(proj\_snd(tr_t)) = front(proj\_snd(tr_t')) \wedge last(proj\_fst(tr_t)) = last(proj\_fst(tr_t'))\end{array}\right)\end{array}\right) & [\text{substitution}] \\
= & L(X)\ ;\ \left(\begin{array}{c}(ok \Rightarrow ok') \wedge wait' = wait \wedge state' = state \wedge \\ \left(\begin{array}{c}\exists\, tr_t, tr_t' \bullet f(tr_t, tr_t', tr, tr', ref, ref') \wedge \\ front(tr) = front(tr') \wedge front(ref) = front(ref') \wedge \\ last(tr) = last(tr')\end{array}\right)\end{array}\right) & [\text{predicate calculus}] \\
= & L(X)\ ;\ \left(\ (ok \Rightarrow ok') \wedge wait' = wait \wedge state' = state \wedge tr = tr' \wedge front(ref) = front(ref')\ \right) & [J_{ct}] \\
= & L(X)\ ;\ J & [\mathbf{CSP2_{ct}}] \\
= & \mathbf{CSP2_{ct}}(L(X))
\end{array}
$$

□

L6. $L(\mathbf{CSP3_t}(X)) = \mathbf{CSP3_{ct}}(L(X))$

Proof.

$$
\begin{array}{ll}
& L(\mathbf{CSP3_t}(X)) & [\mathbf{CSP3_t}(2.10)] \\
= & L(\mathit{Skip}_t\ ;\ X) & [\text{Property 3-L6}] \\
= & L(\mathit{Skip}_t)\ ;\ L(X) & [\text{Law 45}] \\
= & \mathit{Skip}_{ct}\ ;\ L(X) & [\mathbf{CSP3_{ct}}] \\
= & \mathbf{CSP3_{ct}}(L(X))
\end{array}
$$

□

L7. $L(\mathbf{CSP4_t}(X)) = \mathbf{CSP4_{ct}}(L(X))$

Proof.

$$
\begin{aligned}
& L(\mathbf{CSP4_t}(X)) && [\mathbf{CSP4_t}(2.11)]\\
=& L(X \,;\, Skip_t) && [\text{Property 3-L6}]\\
=& L(X) \,;\, L(Skip_t) && [\text{Law 45}]\\
=& L(X) \,;\, Skip_{ct} && [\mathbf{CSP4_{ct}}]\\
=& \mathbf{CSP4_{ct}}(L(X))
\end{aligned}
$$

□

L8. $L(\mathbf{CSP5_t}(X)) = \mathbf{CSP5_{ct}}(L(X))$

Proof.

$$
\begin{aligned}
& L(\mathbf{CSP5_t}(X)) && [\mathbf{CSP5_t}(2.12)]\\
=& L(X \;|||\; Skip_t) && [\text{Law 52}]\\
=& L(X) \;|||\; L(Skip_t) && [\text{Law 45}]\\
=& L(X) \;|||\; Skip_{ct} && [\mathbf{CSP5_{ct}}]\\
=& \mathbf{CSP5_{ct}}(L(X))
\end{aligned}
$$

□

C Parallelism

We give the definitions of *TSync*, *Sync* and *MRef* in parallelism. The synchronisation function *TSync* takes two timed traces and a set of events on which the actions should synchronise, and returns the set containing all possible timed traces. Given two timed traces S_1 and S_2, *TSync* is defined as follows:

$$TSync(S_1, S_2, CS) = TSync(S_2, S_1, CS)$$
$$TSync(\langle\rangle, \langle\rangle, CS) = \{\langle\rangle\}$$
$$TSync(\langle t_1\rangle \frown S_1, \langle\rangle, CS) = \{\langle t'\rangle \mid t' \in Sync(t_1, \langle\rangle, CS)\} \frown TSync(S_1, \langle\rangle, CS)$$
$$TSync(\langle t_1\rangle \frown S_1, \langle t_2\rangle \frown S_2, CS) = \{\langle t'\rangle \mid t' \in Sync(t_1, t_2, CS)\} \frown TSync(S_1, S_2, CS)$$

where *Sync* is defined for the synchronisation of two sequences of events,

$$Sync(t_1, t_2, CS) = Sync(t_2, t_1, CS)$$
$$Sync(\langle\rangle, \langle\rangle, CS) = \{\langle\rangle\}$$
$$Sync(\langle e_1\rangle \frown t, \langle\rangle, CS) = \{\} \quad \textit{iff}\ e_1 \in CS$$
$$Sync(\langle e_1\rangle \frown t, \langle\rangle, CS) = \{\langle e_1\rangle\} \frown Sync(t, \langle\rangle, CS) \quad \textit{iff}\ e_1 \notin CS$$
$$Sync(\langle e_1\rangle \frown t_1, \langle e_2\rangle \frown t_2, CS) = \begin{pmatrix} \{\langle e_1\rangle\} \frown Sync(t_1, \langle e_2\rangle \frown t_2, CS) \cup \\ \{\langle e_2\rangle\} \frown Sync(\langle e_1\rangle \frown t_1, t_2, CS) \end{pmatrix} \quad \textit{iff}\ e_1 \notin CS \wedge e_2 \notin CS$$
$$Sync(\langle e_1\rangle \frown t_1, \langle e_2\rangle \frown t_2, CS) = \{\langle e_1\rangle\} \frown Sync(t_1, \langle e_2\rangle \frown t_2, CS) \quad \textit{iff}\ e_1 \notin CS \wedge e_2 \in CS$$
$$Sync(\langle e_1\rangle \frown t_1, \langle e_1\rangle \frown t_2, CS) = \{\langle e_1\rangle\} \frown Sync(t_1, t_2, CS) \quad \textit{iff}\ e_1 \in CS$$
$$Sync(\langle e_1\rangle \frown t_1, \langle e_2\rangle \frown t_2, CS) = \{\} \quad \textit{iff}\ e_1 \in CS \wedge e_2 \in CS \wedge e_1 \neq e_2$$

And the function *MRef* is used to merge the two refusal sequences.

$$MRef(ref_1, ref_2, CS) = MRef(ref_2, ref_1, CS)$$
$$MRef(\langle\rangle, \langle\rangle, CS) = \langle\rangle$$
$$MRef(\langle r_1\rangle \frown ref_1, \langle\rangle, CS) = \langle r_1\rangle \frown MRef(ref_1, \langle\rangle, CS)$$
$$MRef(\langle r_1\rangle \frown ref_1, \langle r_2\rangle \frown ref_2, CS) = \langle((r_1 \cup r_2) \cap CS) \cup ((r_1 \cap r_2) \setminus CS)\rangle \frown MRef(ref_1, ref_2, CS)$$

Lemma 43.

$$Skip^f = \mathbf{R1_{ct}}(\neg\ ok)$$

Proof.

$$\begin{aligned}
&Skip^f \\
=&(\mathbf{R_{ct}}(\mathbf{true} \vdash \neg\ wait' \wedge tr' = tr \wedge state' = state))^f \\
=&(\mathbf{R1_{ct}} \circ \mathbf{R2_{ct}}(\mathbb{I}_{ct} \lhd wait \rhd (\mathbf{true} \vdash \neg\ wait' \wedge tr' = tr \wedge state' = state)))^f && [\mathbb{I}_{ct}] \\
=&(\mathbf{R1_{ct}} \circ \mathbf{R2_{ct}}(((\neg\ ok \wedge RT) \vee (ok' \wedge \mathbb{I})) \lhd wait \rhd (\mathbf{true} \vdash \neg\ wait' \wedge tr' = tr \wedge state' = state)))^f \\
& && [\text{substitution}] \\
=&\mathbf{R1_{ct}} \circ \mathbf{R2_{ct}}((\neg\ ok \wedge RT) \lhd wait \rhd \neg\ ok)) && [\text{relational calculus}] \\
=&\mathbf{R1_{ct}} \circ \mathbf{R2_{ct}}(\neg\ ok) && [\text{property of } \mathbf{R2_{ct}}] \\
=&\mathbf{R1_{ct}}(\neg\ ok)
\end{aligned}$$

□

Lemma 44. $Skip^t = \mathbf{R1_{ct}}(\neg\ ok) \vee (ok \wedge (\mathbb{I} \lhd wait \rhd (\mathbb{I}^{-ref} \wedge front(ref') = front(ref))))$

Proof.

$$\begin{aligned}
&Skip^t \\
=&(\mathbf{R_{ct}}(\mathbf{true} \vdash \neg\ wait' \wedge tr' = tr \wedge state' = state))^t \\
=&(\mathbf{R1_{ct}} \circ \mathbf{R2_{ct}}(\mathbb{I}_{ct} \lhd wait \rhd (\mathbf{true} \vdash \neg\ wait' \wedge tr' = tr \wedge state' = state)))^t && [\mathbb{I}_{ct}] \\
=&(\mathbf{R1_{ct}} \circ \mathbf{R2_{ct}}(((\neg\ ok \wedge RT) \vee (ok' \wedge \mathbb{I})) \lhd wait \rhd (\mathbf{true} \vdash \neg\ wait' \wedge tr' = tr \wedge state' = state)))^t \\
& && [\text{substitution}] \\
=&\mathbf{R1_{ct}} \circ \mathbf{R2_{ct}}((\neg\ ok \wedge RT) \vee \mathbb{I}) \lhd wait \rhd (\neg\ ok \vee (\neg\ wait' \wedge tr' = tr \wedge state' = state))) \\
& && [\text{relational calculus}] \\
=&\mathbf{R1_{ct}} \circ \mathbf{R2_{ct}}(\neg\ ok \vee (\mathbb{I} \lhd wait \rhd (\neg\ wait' \wedge tr' = tr \wedge state' = state)) && [\text{property of } \mathbf{R2_{ct}}] \\
=&\mathbf{R1_{ct}}(\neg\ ok \vee (\mathbb{I} \lhd wait \rhd (\neg\ wait' \wedge tr' = tr \wedge state' = state)) && [\text{relational calculus}] \\
=&\mathbf{R1_{ct}}(\neg\ ok) \vee (ok \wedge (\mathbb{I} \lhd wait \rhd (\mathbb{I}^{-ref} \wedge front(ref') = front(ref))))
\end{aligned}$$

□

Lemma 45.

$$\begin{aligned}
&(((P\,;U0(out\alpha P)) \wedge (Q\,;U1(out\alpha Q)))_{+\{tr,ref\}}\,;\,M_{\|})^f_f = \\
&\quad \exists\, 0.tr, 0, ref, 1, tr, 1, ref \bullet \left(\begin{array}{l} P^f_f[0.tr, 0.ref/tr', ref'] \wedge Q_f[1.tr, 1.ref/tr', ref'] \\ \wedge\ MTR(tr, tr', 0.tr, 1.tr, ref, ref', 0.ref, 1.ref) \end{array} \right)\,;\, \mathbf{R1_{ct}}(\mathbf{true}) \vee \\
&\quad \exists\, 0.tr, 0, ref, 1, tr, 1, ref \bullet \left(\begin{array}{l} P_f[0.tr, 0.ref/tr', ref'] \wedge Q^f_f[1.tr, 1.ref/tr', ref'] \\ \wedge\ MTR(tr, tr', 0.tr, 1.tr, ref, ref', 0.ref, 1.ref) \end{array} \right)\,;\, \mathbf{R1_{ct}}(\mathbf{true})
\end{aligned}$$

Proof.

$$
\begin{aligned}
&(((P\,;U0(out\alpha P)) \wedge (Q\,;U1(out\alpha Q)))_{+\{tr,ref\}}\,;\, M_{\|})^{f}_{f} && [M_{\|}]\\
=&(((P\,;U0(out\alpha P)) \wedge (Q\,;U1(out\alpha Q)))_{+\{tr,ref\}}\,;\, M_{ct};\ Skip)^{f}_{f}\\
=&(((P_f\,;U0(out\alpha P)) \wedge (Q_f\,;U1(out\alpha Q)))_{+\{tr,ref\}}\,;\, M_{ct};\ Skip^{f}) && [M_{ct}]\\
=&((P_f\,;U0(out\alpha P)) \wedge (Q_f\,;U1(out\alpha Q)))_{+\{tr,ref\}}; && \text{[Lemma 43]}\\
&\left(\begin{array}{l} ok' = (0.ok \wedge 1.ok) \wedge MTR(tr, tr', 0.tr, 1.tr, ref, ref', 0.ref, 1.ref) \wedge \\ wait' = (0.wait \vee 1.wait) \wedge state' = (0.state - s_2) \oplus (1.state - s_1) \end{array}\right);\ Skip^{f}\\
=&((P_f\,;U0(out\alpha P)) \wedge (Q_f\,;U1(out\alpha Q)))_{+\{tr,ref\}}; && \text{[relational calculus]}\\
&\left(\begin{array}{l} ok' = (0.ok \wedge 1.ok) \wedge MTR(tr, tr', 0.tr, 1.tr, ref, ref', 0.ref, 1.ref) \wedge \\ wait' = (0.wait \vee 1.wait) \wedge state' = (0.state - s_2) \oplus (1.state - s_1) \end{array}\right);\ \mathbf{R1_{ct}}(\neg\ ok)\\
=&((P_f\,;U0(out\alpha P)) \wedge (Q_f\,;U1(out\alpha Q)))_{+\{tr,ref\}}; && \text{[relational calculus]}\\
&(\mathbf{false} = (0.ok \wedge 1.ok) \wedge MTR(tr, tr', 0.tr, 1.tr, ref, ref', 0.ref, 1.ref));\ \mathbf{R1_{ct}}(\mathbf{true})\\
=&\left(\begin{array}{l} ((P^{f}_{f}\,;U0(out\alpha P)) \wedge (Q_f\,;U1(out\alpha Q)))_{+\{tr,ref\}}; \\ MTR(tr, tr', 0.tr, 1.tr, ref, ref', 0.ref, 1.ref);\ \mathbf{R1_{ct}}(\mathbf{true}) \end{array}\right) \vee && \text{[relational calculus]}\\
&\left(\begin{array}{l} ((P_f\,;U0(out\alpha P)) \wedge (Q^{f}_{f}\,;U1(out\alpha Q)))_{+\{tr,ref\}}; \\ MTR(tr, tr', 0.tr, 1.tr, ref, ref', 0.ref, 1.ref);\ \mathbf{R1_{ct}}(\mathbf{true}) \end{array}\right)\\
=&\exists\, 0.tr, 0, ref, 1, tr, 1, ref \bullet \left(\begin{array}{l} P^{f}_{f}[0.tr, 0.ref/tr', ref'] \wedge Q_f[1.tr, 1.ref/tr', ref'] \\ \wedge\ MTR(tr, tr', 0.tr, 1.tr, ref, ref', 0.ref, 1.ref) \end{array}\right);\ \mathbf{R1_{ct}}(\mathbf{true}) \vee\\
&\exists\, 0.tr, 0, ref, 1, tr, 1, ref \bullet \left(\begin{array}{l} P_f[0.tr, 0.ref/tr', ref'] \wedge Q^{f}_{f}[1.tr, 1.ref/tr', ref'] \\ \wedge\ MTR(tr, tr', 0.tr, 1.tr, ref, ref', 0.ref, 1.ref) \end{array}\right);\ \mathbf{R1_{ct}}(\mathbf{true})
\end{aligned}
$$

□

Lemma 46.

$$
\begin{aligned}
&(((P\,;U0(out\alpha P)) \wedge (Q\,;U1(out\alpha Q)))_{+\{tr,ref\}}\,;\, M_{\|})^{t}_{f} =\\
&\qquad\left(\begin{array}{l} (((P\,;U0(out\alpha P)) \wedge (Q\,;U1(out\alpha Q)))_{+\{tr,ref\}}\,;\, M_{\|})^{f}_{f} \vee \\ (((P^{t}_{f}\,;U0(out\alpha P)) \wedge (Q^{t}_{f}\,;U1(out\alpha Q)))_{+\{tr,ref\}};\ M_{ct}) \end{array}\right)
\end{aligned}
$$

Proof.

$$
\begin{array}{ll}
& (((P\,;U0(out\alpha P)) \wedge (Q\,;U1(out\alpha Q)))_{+\{tr,ref\}}\,;\,M_{\|})^t_f & [M_{\|}]\\
=& (((P\,;U0(out\alpha P)) \wedge (Q\,;U1(out\alpha Q)))_{+\{tr,ref\}}\,;\,M_{ct};\ Skip)^t_f & \\
=& (((P_f\,;U0(out\alpha P)) \wedge (Q_f\,;U1(out\alpha Q)))_{+\{tr,ref\}}\,;\,M_{ct};\ Skip^t) & [M_{ct}]\\
=& ((P_f\,;U0(out\alpha P)) \wedge (Q_f\,;U1(out\alpha Q)))_{+\{tr,ref\}}; & [\text{Lemma 44}]\\
& \left(\begin{array}{l} ok' = (0.ok \wedge 1.ok) \wedge MTR(tr, tr', 0.tr, 1.tr, ref, ref', 0.ref, 1.ref) \wedge \\ wait' = (0.wait \vee 1.wait) \wedge state' = (0.state - s_2) \oplus (1.state - s_1) \end{array}\right);\\
& \mathbf{R1_{ct}}(\neg\ ok) \vee (ok \wedge (\mathbb{I} \lhd wait \rhd (\mathbb{I}^{-ref} \wedge front(ref') = front(ref))))\\
=& \left(\begin{array}{l} ((P_f\,;U0(out\alpha P)) \wedge (Q_f\,;U1(out\alpha Q)))_{+\{tr,ref\}}; \\ \left(\begin{array}{l} ok' = (0.ok \wedge 1.ok) \wedge MTR(tr, tr', 0.tr, 1.tr, ref, ref', 0.ref, 1.ref) \wedge \\ wait' = (0.wait \vee 1.wait) \wedge state' = (0.state - s_2) \oplus (1.state - s_1) \end{array}\right);\ \mathbf{R1_{ct}}(\neg\ ok) \end{array}\right) \vee\\
& \left(\begin{array}{l} ((P_f\,;U0(out\alpha P)) \wedge (Q_f\,;U1(out\alpha Q)))_{+\{tr,ref\}}; \\ \left(\begin{array}{l} ok' = (0.ok \wedge 1.ok) \wedge MTR(tr, tr', 0.tr, 1.tr, ref, ref', 0.ref, 1.ref) \wedge \\ wait' = (0.wait \vee 1.wait) \wedge state' = (0.state - s_2) \oplus (1.state - s_1) \end{array}\right); \\ (ok \wedge (\mathbb{I} \lhd wait \rhd (\mathbb{I}^{-ref} \wedge front(ref') = front(ref)))) \end{array}\right) & [\text{Lemma 18}]\\
=& (((P\,;U0(out\alpha P)) \wedge (Q\,;U1(out\alpha Q)))_{+\{tr,ref\}}\,;\,M_{\|})^f_f \vee & [\text{relational calculus}]\\
& \left(\begin{array}{l} ((P_f\,;U0(out\alpha P)) \wedge (Q_f\,;U1(out\alpha Q)))_{+\{tr,ref\}}; \\ \left(\begin{array}{l} ok' = (0.ok \wedge 1.ok) \wedge MTR(tr, tr', 0.tr, 1.tr, ref, ref', 0.ref, 1.ref) \wedge \\ wait' = (0.wait \vee 1.wait) \wedge state' = (0.state - s_2) \oplus (1.state - s_1) \end{array}\right); \\ (ok \wedge (\mathbb{I} \lhd wait \rhd (\mathbb{I}^{-ref} \wedge front(ref') = front(ref)))) \end{array}\right) & [\text{Lemma 18}]\\
=& (((P\,;U0(out\alpha P)) \wedge (Q\,;U1(out\alpha Q)))_{+\{tr,ref\}}\,;\,M_{\|})^f_f \vee & [\text{relational calculus}]\\
& \left(\begin{array}{l} ((P^t_f\,;U0(out\alpha P)) \wedge (Q^t_f\,;U1(out\alpha Q)))_{+\{tr,ref\}}; \\ \left(\begin{array}{l} ok' = (0.ok \wedge 1.ok) \wedge MTR(tr, tr', 0.tr, 1.tr, ref, ref', 0.ref, 1.ref) \wedge \\ wait' = (0.wait \vee 1.wait) \wedge state' = (0.state - s_2) \oplus (1.state - s_1) \end{array}\right); \\ (\mathbb{I} \lhd wait \rhd (\mathbb{I}^{-ref} \wedge front(ref') = front(ref))) \end{array}\right) & [M_{ct}]\\
=& (((P\,;U0(out\alpha P)) \wedge (Q\,;U1(out\alpha Q)))_{+\{tr,ref\}}\,;\,M_{\|})^f_f \vee & [P \text{ and } Q \text{ are } \mathbf{CSP4_{ct}}]\\
& \left(\left(\begin{array}{l} (P^t_f\,;U0(out\alpha P)) \wedge \\ (Q^t_f\,;U1(out\alpha Q)) \end{array}\right)_{+\{tr,ref\}};\ M_{ct};\ (\mathbb{I} \lhd wait \rhd (\mathbb{I}^{-ref} \wedge front(ref') = front(ref)))\right)\\
=& \left(\begin{array}{l} (((P\,;U0(out\alpha P)) \wedge (Q\,;U1(out\alpha Q)))_{+\{tr,ref\}}\,;\,M_{\|})^f_f \vee \\ (((P^t_f\,;U0(out\alpha P)) \wedge (Q^t_f\,;U1(out\alpha Q)))_{+\{tr,ref\}};\ M_{ct}) \end{array}\right) &
\end{array}
$$

□

Theorem 31.

$$
\begin{array}{l}
(((P\,;U0(out\alpha P)) \wedge (Q\,;U1(out\alpha Q)))_{+\{tr,ref\}}\,;\,M_{\|}) =\\
\left(\begin{array}{c} \exists 0.tr, 0, ref, 1, tr, 1, ref \bullet \left(\begin{array}{l} P^f_f[0.tr, 0.ref/tr', ref'] \wedge Q_f[1.tr, 1.ref/tr', ref'] \\ \wedge\ MTR(tr, tr', 0.tr, 1.tr, ref, ref', 0.ref, 1.ref) \end{array}\right);\ \mathbf{R1_{ct}}(\mathbf{true}) \vee \\ \exists 0.tr, 0, ref, 1, tr, 1, ref \bullet \left(\begin{array}{l} P_f[0.tr, 0.ref/tr', ref'] \wedge Q^f_f[1.tr, 1.ref/tr', ref'] \\ \wedge\ MTR(tr, tr', 0.tr, 1.tr, ref, ref', 0.ref, 1.ref) \end{array}\right);\ \mathbf{R1_{ct}}(\mathbf{true}) \\ \vdash \\ (((P^t_f\,;U0(out\alpha P)) \wedge (Q^t_f\,;U1(out\alpha Q)))_{+\{tr,ref\}};\ M_{ct}) \end{array}\right)
\end{array}
$$

Proof. The proof is simply based on Lemma 18 and Lemma 19. □

D Deadline

Law 83. *Wait* d ; *Miracle* $= \mathbf{R_{ct}}(\mathbf{true} \vdash \frown\!/tr' = \frown\!/tr \wedge \#tr' - \#tr < d \wedge wait')$

Proof.

$$
\begin{array}{ll}
& \mathit{Wait}\ d\,;\ \mathit{Miracle} \qquad [\text{defs of } \mathit{Wait} \text{ and } \mathit{Miracle}] \\
= & \mathbf{R_{ct}}\Big(\mathbf{true} \vdash \frown\!/tr' = \frown\!/tr \wedge \Big(\#tr' - \#tr < d \lhd \mathit{wait}' \rhd \begin{pmatrix} \#tr' - \#tr = d \\ \wedge\ \mathit{state}' = \mathit{state} \end{pmatrix}\Big)\Big)\ ;\ \mathbf{R_{ct}}(\mathbf{true} \vdash \mathbf{false}) \\
& \qquad [\text{Theorem 6}] \\
= & \mathbf{R_{ct}}\left(\begin{array}{c} \neg\,(\mathbf{R1_{ct}}(\mathbf{false})\,;\,\mathbf{R1_{ct}}(\mathbf{true})) \wedge \neg\,(\mathbf{R1_{ct}}((\mathit{Wait}\ d)^t_f)\,;\,\mathbf{R1_{ct}}(\neg\,\mathit{wait} \wedge \mathbf{R2_{ct}}(\mathbf{false}))) \\ \vdash \\ \mathbf{R1_{ct}}\left(\frown\!/tr' = \frown\!/tr \wedge \left(\begin{array}{c} \#tr' - \#tr < d \\ \lhd\ \mathit{wait}'\ \rhd \\ \begin{pmatrix} \#tr' - \#tr = d \\ \wedge\ \mathit{state}' = \mathit{state} \end{pmatrix} \end{array}\right)\right)\ ;\ \mathbf{R1_{ct}}(\mathbb{I} \lhd \mathit{wait} \rhd \mathbf{R2_{ct}}(\mathbf{false})) \end{array}\right) \qquad [\text{rel. cal.}] \\
= & \mathbf{R_{ct}}\left(\mathbf{true} \vdash \mathbf{R_{ct}}\left(\frown\!/tr' = \frown\!/tr \wedge \left(\begin{array}{c} \#tr' - \#tr < d \\ \lhd\ \mathit{wait}'\ \rhd \\ \begin{pmatrix} \#tr' - \#tr = d \\ \wedge\ \mathit{state}' = \mathit{state} \end{pmatrix} \end{array}\right)\right)\ ;\ \mathbf{R1_{ct}}(\mathbb{I} \lhd \mathit{wait} \rhd \mathbf{R2_{ct}}(\mathbf{false}))\right) \\
& \qquad [\mathbf{R2_{ct}}(\mathbf{false}) = \mathbf{false} \text{ and relational calculus}] \\
= & \mathbf{R_{ct}}\left(\mathbf{true} \vdash \mathbf{R1_{ct}}\left(\frown\!/tr' = \frown\!/tr \wedge \left(\begin{array}{c} \#tr' - \#tr < d \\ \lhd\ \mathit{wait}'\ \rhd \\ \begin{pmatrix} \#tr' - \#tr = d \\ \wedge\ \mathit{state}' = \mathit{state} \end{pmatrix} \end{array}\right)\right)\ ;\ \mathbf{R1_{ct}}(\mathbb{I} \wedge \mathit{wait})\right) \qquad [\text{relational calculus}] \\
= & \mathbf{R_{ct}}(\mathbf{true} \vdash \mathbf{R1_{ct}}(\frown\!/tr' = \frown\!/tr \wedge \#tr' - \#tr < d \wedge \mathit{wait}')\,;\,\mathbf{R1_{ct}}(\mathbb{I} \wedge \mathit{wait})) \qquad [\mathbf{R1_{ct}} \text{ and unit law}] \\
= & \mathbf{R_{ct}}(\mathbf{true} \vdash \frown\!/tr' = \frown\!/tr \wedge \#tr' - \#tr < d \wedge \mathit{wait}')
\end{array}
$$

□

Theorem 23

$$
d \blacktriangleleft P = \mathbf{R_{ct}}\left(\begin{array}{c} \neg\,((P^f_f \wedge tr' = tr)\,;\,\mathbf{R1_{ct}}(\mathbf{true})) \wedge \neg\,(P^f_f \wedge \mathit{head}(\mathit{diff}(tr', tr)) \neq \langle\rangle) \wedge \\ \neg\,((P^f_f \wedge \#tr' - \#tr \leq d \wedge \frown\!/tr' = \frown\!/tr);\,\mathbf{R1_{ct}}(\mathbf{true})) \wedge \\ \neg\,(P^f_f \wedge ((\#tr' - \#tr < d \wedge \frown\!/tr' = \frown\!/tr \wedge \mathit{wait}')\,;\,\mathit{head}(tr' - tr) \neq \langle\rangle)) \\ \vdash \left(\begin{array}{c} (P^t_f \wedge \#tr' - \#tr < d \wedge \mathit{wait}' \wedge \frown\!/tr' = \frown\!/tr) \vee \\ (P^t_f \wedge (((\#tr' - \#tr < d \wedge \frown tr' = \frown tr \wedge \mathit{wait}');\,\mathit{term\_next}) \vee \mathit{term\_now})) \end{array}\right) \end{array}\right)
$$

Proof.

$$
\begin{array}{ll}
& d \blacktriangleleft P \qquad [\text{def of } \blacktriangleleft] \\
= & P \,\Box\, (\mathit{Wait}\ d\,;\ \mathit{Miracle}) \qquad [\Box\text{-Theorem 15}] \\
= & \mathbf{R_{ct}}\left(\begin{array}{c} \neg\,(((P^f_f \vee (\mathit{Wait}\ d;\ \mathit{Miracle})^f_f) \wedge tr' = tr)\,;\,\mathbf{R1_{ct}}(\mathbf{true})) \wedge \\ \neg\,((P^f_f \wedge (((\mathit{Wait}\ d;\ \mathit{Miracle})_f \wedge \frown\!/tr' = \frown\!/tr \wedge \mathit{wait}');\ tr' - tr = \langle\langle\rangle\rangle));\,\mathbf{R1_{ct}}(\mathbf{true})) \wedge \\ \neg\,(((\mathit{Wait}\ d;\ \mathit{Miracle})^f_f \wedge ((P_f \wedge \frown\!/tr' = \frown\!/tr \wedge \mathit{wait}');\ tr' - tr = \langle\langle\rangle\rangle));\,\mathbf{R1_{ct}}(\mathbf{true})) \wedge \\ \neg\,(P^f_f \wedge (((\mathit{Wait}\ d;\ \mathit{Miracle})_f \wedge \frown\!/tr' = \frown\!/tr \wedge \mathit{wait}')\,;\,\mathit{head}(tr' - tr) \neq \langle\rangle)) \wedge \\ \neg\,((\mathit{Wait}\ d;\ \mathit{Miracle})^f_f \wedge ((P_f \wedge \frown\!/tr' = \frown\!/tr \wedge \mathit{wait}')\,;\,\mathit{head}(tr' - tr) \neq \langle\rangle)) \wedge \\ \neg\,((P^f_f \vee (\mathit{Wait}\ d;\ \mathit{Miracle})^f_f) \wedge \mathit{head}(\mathit{diff}(tr', tr)) \neq \langle\rangle) \\ \vdash \left(\begin{array}{c} (P^t_f \wedge (\mathit{Wait}\ d;\ \mathit{Miracle})^t_f \wedge \mathit{wait}' \wedge \frown\!/tr' = \frown\!/tr) \vee \\ (\mathit{Diff}(P^t_f, (\mathit{Wait}\ d;\ \mathit{Miracle})^t_f) \wedge (P^t_f \vee (\mathit{Wait}\ d;\ \mathit{Miracle})^t_f)) \end{array}\right) \end{array}\right) \\
& \qquad [\text{Law 83 and its precondition}]
\end{array}
$$

$$= \mathbf{R_{ct}} \left(\begin{array}{c} \neg\,(((P_f^f \vee \mathbf{false}) \wedge tr' = tr)\,;\, \mathbf{R1_{ct}}(\mathbf{true})) \wedge \\ \neg\,((P_f^f \wedge (((\mathbf{false} \vee (Wait\ d;\ Miracle)_f^t) \wedge \frown\!/tr' = \frown\!/tr \wedge wait');\ tr' - tr = \langle\langle\rangle\rangle);\ \mathbf{R1_{ct}}(\mathbf{true})) \wedge \\ \neg\,((\mathbf{false} \wedge ((P_f \wedge \frown\!/tr' = \frown\!/tr \wedge wait');\ tr' - tr = \langle\langle\rangle\rangle));\ \mathbf{R1_{ct}}(\mathbf{true})) \wedge \\ \neg\,(P_f^f \wedge (((\mathbf{false} \vee (Wait\ d;\ Miracle)_f^t) \wedge \frown\!/tr' = \frown\!/tr \wedge wait')\,;\ head(tr' - tr) \neq \langle\rangle)) \wedge \\ \neg\,((\mathbf{false} \wedge ((P_f \wedge \frown\!/tr' = \frown\!/tr \wedge wait')\,;\ head(tr' - tr) \neq \langle\rangle)) \wedge \\ \neg\,((P_f^f \vee \mathbf{false}) \wedge head(diff(tr', tr)) \neq \langle\rangle) \\ \vdash \left(\begin{array}{c} (P_f^t \wedge (Wait\ d;\ Miracle)_f^t \wedge wait' \wedge \frown\!/tr' = \frown\!/tr) \vee \\ (Diff(P_f^t, (Wait\ d;\ Miracle)_f^t) \wedge (P_f^t \vee (Wait\ d;\ Miracle)_f^t)) \end{array} \right) \end{array} \right)$$

[relational calculus]

$$= \mathbf{R_{ct}} \left(\begin{array}{c} \neg\,((P_f^f \wedge tr' = tr)\,;\, \mathbf{R1_{ct}}(\mathbf{true})) \wedge \neg\,(P_f^f \wedge head(diff(tr', tr)) \neq \langle\rangle) \wedge \\ \neg\,((P_f^f \wedge (((Wait\ d;\ Miracle)_f^t \wedge \frown\!/tr' = \frown\!/tr \wedge wait');\ tr' - tr = \langle\langle\rangle\rangle);\ \mathbf{R1_{ct}}(\mathbf{true})) \wedge \\ \neg\,(P_f^f \wedge (((Wait\ d;\ Miracle)_f^t \wedge \frown\!/tr' = \frown\!/tr \wedge wait')\,;\ head(tr' - tr) \neq \langle\rangle)) \\ \vdash \left(\begin{array}{c} (P_f^t \wedge (Wait\ d;\ Miracle)_f^t \wedge wait' \wedge \frown\!/tr' = \frown\!/tr) \vee \\ (Diff(P_f^t, (Wait\ d;\ Miracle)_f^t) \wedge (P_f^t \vee (Wait\ d;\ Miracle)_f^t)) \end{array} \right) \end{array} \right)$$

[Law 83 and the postcondition]

$$= \mathbf{R_{ct}} \left(\begin{array}{c} \neg\,((P_f^f \wedge tr' = tr)\,;\, \mathbf{R1_{ct}}(\mathbf{true})) \wedge \neg\,(P_f^f \wedge head(diff(tr', tr)) \neq \langle\rangle) \wedge \\ \neg\,((P_f^f \wedge ((\#tr' - \#tr < d \wedge \frown\!/tr' = \frown\!/tr \wedge wait');\ tr' - tr = \langle\langle\rangle\rangle);\ \mathbf{R1_{ct}}(\mathbf{true})) \wedge \\ \neg\,(P_f^f \wedge ((\#tr' - \#tr < d \wedge \frown\!/tr' = \frown\!/tr \wedge wait')\,;\ head(tr' - tr) \neq \langle\rangle)) \\ \vdash \left(\begin{array}{c} (P_f^t \wedge \#tr' - \#tr < d \wedge wait' \wedge \frown\!/tr' = \frown\!/tr) \vee \\ (Diff(P_f^t, (Wait\ d;\ Miracle)_f^t) \wedge (P_f^t \vee (Wait\ d;\ Miracle)_f^t)) \end{array} \right) \end{array} \right)$$

[rel. cal.]

$$= \mathbf{R_{ct}} \left(\begin{array}{c} \neg\,((P_f^f \wedge tr' = tr)\,;\, \mathbf{R1_{ct}}(\mathbf{true})) \wedge \neg\,(P_f^f \wedge head(diff(tr', tr)) \neq \langle\rangle) \wedge \\ \neg\,((P_f^f \wedge \#tr' - \#tr \leq d \wedge \frown\!/tr' = \frown\!/tr);\ \mathbf{R1_{ct}}(\mathbf{true})) \wedge \\ \neg\,(P_f^f \wedge ((\#tr' - \#tr < d \wedge \frown\!/tr' = \frown\!/tr \wedge wait')\,;\ head(tr' - tr) \neq \langle\rangle)) \\ \vdash \left(\begin{array}{c} (P_f^t \wedge \#tr' - \#tr < d \wedge wait' \wedge \frown\!/tr' = \frown\!/tr) \vee \\ (Diff(P_f^t, (Wait\ d;\ Miracle)_f^t) \wedge (P_f^t \vee (Wait\ d;\ Miracle)_f^t)) \end{array} \right) \end{array} \right)$$

[*Diff* (3.40),Law 83 and the postcondition]

$$= \mathbf{R_{ct}} \left(\begin{array}{c} \neg\,((P_f^f \wedge tr' = tr)\,;\, \mathbf{R1_{ct}}(\mathbf{true})) \wedge \neg\,(P_f^f \wedge head(diff(tr', tr)) \neq \langle\rangle) \wedge \\ \neg\,((P_f^f \wedge \#tr' - \#tr \leq d \wedge \frown\!/tr' = \frown\!/tr);\ \mathbf{R1_{ct}}(\mathbf{true})) \wedge \\ \neg\,(P_f^f \wedge ((\#tr' - \#tr < d \wedge \frown\!/tr' = \frown\!/tr \wedge wait')\,;\ head(tr' - tr) \neq \langle\rangle)) \\ \vdash \left(\begin{array}{c} (P_f^t \wedge \#tr' - \#tr < d \wedge wait' \wedge \frown\!/tr' = \frown\!/tr) \vee \\ \left(\begin{array}{c} (((P_f^t \wedge \#tr' - \#tr < d \wedge \frown tr' = \frown tr \wedge wait');\ term\_next) \vee term\_now) \\ \wedge\ (P_f^t \vee (\#tr' - \#tr < d \wedge \frown tr' = \frown tr \wedge wait')) \end{array} \right) \end{array} \right) \end{array} \right)$$

[*term_now*(4.61), *term_next*(4.62) and relational calculus]

$$= \mathbf{R_{ct}} \left(\begin{array}{c} \neg\,((P_f^f \wedge tr' = tr)\,;\, \mathbf{R1_{ct}}(\mathbf{true})) \wedge \neg\,(P_f^f \wedge head(diff(tr', tr)) \neq \langle\rangle) \wedge \\ \neg\,((P_f^f \wedge \#tr' - \#tr \leq d \wedge \frown\!/tr' = \frown\!/tr);\ \mathbf{R1_{ct}}(\mathbf{true})) \wedge \\ \neg\,(P_f^f \wedge ((\#tr' - \#tr < d \wedge \frown\!/tr' = \frown\!/tr \wedge wait')\,;\ head(tr' - tr) \neq \langle\rangle)) \\ \vdash \\ (P_f^t \wedge \#tr' - \#tr < d \wedge wait' \wedge \frown\!/tr' = \frown\!/tr) \vee \\ (P_f^t \wedge (((P_f^t \wedge \#tr' - \#tr < d \wedge \frown tr' = \frown tr \wedge wait');\ term\_next) \vee term\_now)) \end{array} \right)$$

[rel. cal.]

$$= \mathbf{R_{ct}} \left(\begin{array}{c} \neg\left((P_f^f \wedge tr' = tr)\,;\, \mathbf{R1_{ct}}(\mathbf{true})\right) \wedge \neg\left(P_f^f \wedge head(diff(tr', tr)) \neq \langle\rangle\right) \wedge \\ \neg\left((P_f^f \wedge \#tr' - \#tr \leq d \wedge \frown/tr' = \frown/tr);\, \mathbf{R1_{ct}}(\mathbf{true})\right) \wedge \\ \neg\left(P_f^f \wedge ((\#tr' - \#tr < d \wedge \frown/tr' = \frown/tr \wedge wait')\,;\, head(tr' - tr) \neq \langle\rangle)\right) \\ \vdash \\ (P_f^t \wedge \#tr' - \#tr < d \wedge wait' \wedge \frown/tr' = \frown/tr) \vee \\ (P_f^t \wedge (((\#tr' - \#tr < d \wedge \frown tr' = \frown tr \wedge wait');\, term\_next) \vee term\_now)) \end{array} \right)$$

□

E Interrupts

E.1 Genetic interrupts

Lemma 47. $\frown/1.tr = \frown/tr \Rightarrow disable(tr, 0.tr, 1.tr)$

Proof. The proof simply relies on unfolding the definition of *disable*. □

Lemma 48.

$$\begin{array}{l} \frown/1.tr = \frown/tr \wedge \#0.tr = \#0.ref \wedge 1.tr = \#1.ref \Rightarrow \\ (IMTR(tr, tr', 0.tr, 1.tr, ref, ref', 0.ref, 1.ref) = (tr' = 0.tr \wedge ref' = 0.ref)) \end{array}$$

Proof. The proof is based on the induction law. □

Law 72 $(P \triangle Stop) = P = (Stop \triangle P)$

Proof.

$(P \triangle Stop)$ [Definition 18]

$$= \mathbf{R_{ct}} \left(\begin{array}{c} \neg\left(((P_f^f \vee Stop_f^f) \wedge tr' = tr)\,;\, \mathbf{R1_{ct}}(\mathbf{true})\right) \wedge \\ \neg\ \exists \left(\begin{array}{c} 0.tr, 0.ref, \\ 1.tr, 1.ref \end{array} \right) \bullet \left(\begin{array}{c} P_f^f[0.tr, 0.ref/tr', ref'] \wedge \\ Stop_f^t[1.tr, 1.ref/tr', ref'] \wedge \\ disable(tr, 0.tr, 1.tr) \wedge \\ IMTR\left(\begin{array}{c} tr, tr', 0.tr, 1.tr, \\ ref, ref', 0.ref, 1.ref \end{array} \right) \end{array} \right) ;\mathbf{R1_{ct}}(\mathbf{true}) \wedge \\ \neg\ \exists \left(\begin{array}{c} 0.tr, 0.ref, \\ 1.tr, 1.ref \end{array} \right) \bullet \left(\begin{array}{c} P_f^t[0.tr, 0.ref/tr', ref'] \wedge \\ Stop_f^f[1.tr, 1.ref/tr', ref'] \wedge \\ enable(tr, 0.tr, 1.tr) \wedge \\ IMTR\left(\begin{array}{c} tr, tr', 0.tr, 1.tr, \\ ref, ref', 0.ref, 1.ref \end{array} \right) \end{array} \right) ;\mathbf{R1_{ct}}(\mathbf{true}) \\ \vdash \\ ((P_f^t\,;\, U0(out\alpha P)) \wedge (Stop_f^t\,;\, U1(out\alpha Stop)))_{+\{tr,ref\}}\,;\, (IM1 \vee IM2) \end{array} \right)$$

[*Stop* and its precondition]

$$= \mathbf{R_{ct}} \left(\begin{array}{c} \neg\left(((P_f^f \vee \mathbf{false}) \wedge tr' = tr)\,;\, \mathbf{R1_{ct}}(\mathbf{true})\right) \wedge \\ \neg\ \exists \left(\begin{array}{c} 0.tr, 0.ref, \\ 1.tr, 1.ref \end{array} \right) \bullet \left(\begin{array}{c} P_f^f[0.tr, 0.ref/tr', ref'] \wedge \\ Stop_f^t[1.tr, 1.ref/tr', ref'] \wedge \\ disable(tr, 0.tr, 1.tr) \wedge \\ IMTR\left(\begin{array}{c} tr, tr', 0.tr, 1.tr, \\ ref, ref', 0.ref, 1.ref \end{array} \right) \end{array} \right) ;\mathbf{R1_{ct}}(\mathbf{true}) \wedge \\ \neg\ \exists \left(\begin{array}{c} 0.tr, 0.ref, \\ 1.tr, 1.ref \end{array} \right) \bullet \left(\begin{array}{c} P_f^t[0.tr, 0.ref/tr', ref'] \wedge \\ \mathbf{false}[1.tr, 1.ref/tr', ref'] \wedge \\ enable(tr, 0.tr, 1.tr) \wedge \\ IMTR\left(\begin{array}{c} tr, tr', 0.tr, 1.tr, \\ ref, ref', 0.ref, 1.ref \end{array} \right) \end{array} \right) ;\mathbf{R1_{ct}}(\mathbf{true}) \\ \vdash \\ ((P_f^t\,;\, U0(out\alpha P)) \wedge (Stop_f^t\,;\, U1(out\alpha Stop)))_{+\{tr,ref\}}\,;\, (IM1 \vee IM2) \end{array} \right)$$

[rel. calculus]

$$
=\mathbf{R_{ct}}\left(\begin{array}{c}
\neg\left((P_f^f \wedge tr'=tr)\,;\,\mathbf{R1_{ct}}(\mathbf{true})\right)\wedge \\
\neg\ \exists\left(\begin{array}{c}0.tr, 0.ref,\\ 1.tr, 1.ref\end{array}\right)\bullet\left(\begin{array}{c}P_f^f[0.tr, 0.ref/tr', ref']\wedge\\ Stop_f^t[1.tr, 1.ref/tr', ref']\wedge\\ disable(tr, 0.tr, 1.tr)\wedge\\ IMTR\left(\begin{array}{c}tr, tr', 0.tr, 1.tr,\\ ref, ref', 0.ref, 1.ref\end{array}\right)\end{array}\right);\mathbf{R1_{ct}}(\mathbf{true})\wedge\mathbf{true}\\
\vdash\\
((P_f^t\,;\,U0(out\alpha P))\wedge(Stop_f^t\,;\,U1(out\alpha Stop)))_{+\{tr,ref\}}\,;\,(IM1\vee IM2)
\end{array}\right)
$$

[*Stop* and its postcondition]

$$
=\mathbf{R_{ct}}\left(\begin{array}{c}
\neg\left((P_f^f \wedge tr'=tr)\,;\,\mathbf{R1_{ct}}(\mathbf{true})\right)\wedge \\
\neg\ \exists\left(\begin{array}{c}0.tr, 0.ref,\\ 1.tr, 1.ref\end{array}\right)\bullet\left(\begin{array}{c}P_f^f[0.tr, 0.ref/tr', ref']\wedge\\ (wait'\wedge{}^\frown/tr'={}^\frown/tr)[1.tr, 1.ref/tr', ref']\wedge\\ disable(tr, 0.tr, 1.tr)\wedge\\ IMTR\left(\begin{array}{c}tr, tr', 0.tr, 1.tr,\\ ref, ref', 0.ref, 1.ref\end{array}\right)\end{array}\right);\mathbf{R1_{ct}}(\mathbf{true})\\
\vdash\\
((P_f^t\,;\,U0(out\alpha P))\wedge((wait'\wedge{}^\frown/tr'={}^\frown/tr)\,;\,U1(out\alpha Stop)))_{+\{tr,ref\}}\,;\,(IM1\vee IM2)
\end{array}\right)
$$

[Lemma 47,48]

$$
=\mathbf{R_{ct}}\left(\begin{array}{c}
\neg\left((P_f^f \wedge tr'=tr)\,;\,\mathbf{R1_{ct}}(\mathbf{true})\right)\wedge \\
\neg\ \exists\left(\begin{array}{c}0.tr, 0.ref,\\ 1.tr, 1.ref\end{array}\right)\bullet\left(\begin{array}{c}P_f^f[0.tr, 0.ref/tr', ref']\wedge\\ (wait'\wedge{}^\frown/tr'={}^\frown/tr)[1.tr, 1.ref/tr', ref']\wedge\\ tr'=0.tr\wedge ref'=0.ref\end{array}\right);\mathbf{R1_{ct}}(\mathbf{true})\\
\vdash\\
((P_f^t\,;\,U0(out\alpha P))\wedge((wait'\wedge{}^\frown/tr'={}^\frown/tr)\,;\,U1(out\alpha Stop)))_{+\{tr,ref\}}\,;\,(IM1\vee IM2)
\end{array}\right)
$$

[predicate calculus]

$$
=\mathbf{R_{ct}}\left(\begin{array}{c}
\neg\left((P_f^f \wedge tr'=tr)\,;\,\mathbf{R1_{ct}}(\mathbf{true})\right)\wedge\neg\left(P_f^f;\,\mathbf{R1_{ct}}(\mathbf{true})\right)\\
\vdash\\
((P_f^t\,;\,U0(out\alpha P))\wedge((wait'\wedge{}^\frown/tr'={}^\frown/tr)\,;\,U1(out\alpha Stop)))_{+\{tr,ref\}}\,;\,(IM1\vee IM2)
\end{array}\right)
$$

[P is $\mathbf{CSP4_{ct}}$ and relational calculus]

$$
=\mathbf{R_{ct}}\left(\begin{array}{c}
\neg\ P_f^f\vdash\\
((P_f^t\,;\,U0(out\alpha P))\wedge((wait'\wedge{}^\frown/tr'={}^\frown/tr)\,;\,U1(out\alpha Stop)))_{+\{tr,ref\}}\,;\,(IM1\vee IM2)
\end{array}\right)
$$

[Lemma 47 and ${}^\frown/1.tr={}^\frown/tr$ contradicts with $enable(tr, 0.tr, 1.tr)$ in $IM2$]

$$
=\mathbf{R_{ct}}\left(\begin{array}{c}
\neg\ P_f^f\vdash\\
((P_f^t\,;\,U0(out\alpha P))\wedge((wait'\wedge{}^\frown/tr'={}^\frown/tr)\,;\,U1(out\alpha Stop)))_{+\{tr,ref\}}\,;\,IM1
\end{array}\right)
$$

[Lemma 48 and relational calculus]

$$
=\mathbf{R_{ct}}(\neg\ P_f^f\vdash P_f^t)
$$

[Theorem 1]

$$
=P
$$

$P = Stop \triangle P$ can be probed in a similar manner. □

Law 73 $P \triangle Skip = P$

Proof.

$$
\begin{array}{ll}
(P \triangle \mathit{Skip}) & [\text{Definition 18}] \\
= \mathbf{R_{ct}} \left(\begin{array}{c}
\neg \left(((P_f^f \vee \mathit{Skip}_f^f) \wedge tr' = tr) ; \mathbf{R1_{ct}}(\mathbf{true}) \right) \wedge \\
\neg \exists \left(\begin{array}{c} 0.tr, 0.ref, \\ 1.tr, 1.ref \end{array} \right) \bullet \left(\begin{array}{c} P_f^f[0.tr, 0.ref / tr', ref'] \wedge \\ \mathit{Skip}_f^t[1.tr, 1.ref / tr', ref'] \wedge \\ \mathit{disable}(tr, 0.tr, 1.tr) \wedge \\ \mathit{IMTR} \left(\begin{array}{c} tr, tr', 0.tr, 1.tr, \\ ref, ref', 0.ref, 1.ref \end{array} \right) \end{array} \right) ; \mathbf{R1_{ct}}(\mathbf{true}) \wedge \\
\neg \exists \left(\begin{array}{c} 0.tr, 0.ref, \\ 1.tr, 1.ref \end{array} \right) \bullet \left(\begin{array}{c} P_f^t[0.tr, 0.ref / tr', ref'] \wedge \\ \mathit{Skip}_f^f[1.tr, 1.ref / tr', ref'] \wedge \\ \mathit{enable}(tr, 0.tr, 1.tr) \wedge \\ \mathit{IMTR} \left(\begin{array}{c} tr, tr', 0.tr, 1.tr, \\ ref, ref', 0.ref, 1.ref \end{array} \right) \end{array} \right) ; \mathbf{R1_{ct}}(\mathbf{true}) \\
\vdash \\
((P_f^t ; U0(\mathit{out}\alpha P)) \wedge (\mathit{Skip}_f^t ; U1(\mathit{out}\alpha \mathit{Skip})))_{+\{tr, ref\}} ; (IM1 \vee IM2)
\end{array} \right) \\
& [\mathit{Skip} \text{ and its precondition}] \\
= \mathbf{R_{ct}} \left(\begin{array}{c}
\neg \left(((P_f^f \vee \mathbf{false}) \wedge tr' = tr) ; \mathbf{R1_{ct}}(\mathbf{true}) \right) \wedge \\
\neg \exists \left(\begin{array}{c} 0.tr, 0.ref, \\ 1.tr, 1.ref \end{array} \right) \bullet \left(\begin{array}{c} P_f^f[0.tr, 0.ref / tr', ref'] \wedge \\ \mathit{Skip}_f^t[1.tr, 1.ref / tr', ref'] \wedge \\ \mathit{disable}(tr, 0.tr, 1.tr) \wedge \\ \mathit{IMTR} \left(\begin{array}{c} tr, tr', 0.tr, 1.tr, \\ ref, ref', 0.ref, 1.ref \end{array} \right) \end{array} \right) ; \mathbf{R1_{ct}}(\mathbf{true}) \wedge \\
\neg \exists \left(\begin{array}{c} 0.tr, 0.ref, \\ 1.tr, 1.ref \end{array} \right) \bullet \left(\begin{array}{c} P_f^t[0.tr, 0.ref / tr', ref'] \wedge \\ \mathbf{false}[1.tr, 1.ref / tr', ref'] \wedge \\ \mathit{enable}(tr, 0.tr, 1.tr) \wedge \\ \mathit{IMTR} \left(\begin{array}{c} tr, tr', 0.tr, 1.tr, \\ ref, ref', 0.ref, 1.ref \end{array} \right) \end{array} \right) ; \mathbf{R1_{ct}}(\mathbf{true}) \\
\vdash \\
((P_f^t ; U0(\mathit{out}\alpha P)) \wedge (\mathit{Skip}_f^t ; U1(\mathit{out}\alpha \mathit{Stop})))_{+\{tr, ref\}} ; (IM1 \vee IM2)
\end{array} \right) & [\text{rel. calculus}] \\
= \mathbf{R_{ct}} \left(\begin{array}{c}
\neg \left((P_f^f \wedge tr' = tr) ; \mathbf{R1_{ct}}(\mathbf{true}) \right) \wedge \\
\neg \exists \left(\begin{array}{c} 0.tr, 0.ref, \\ 1.tr, 1.ref \end{array} \right) \bullet \left(\begin{array}{c} P_f^f[0.tr, 0.ref / tr', ref'] \wedge \\ \mathit{Skip}_f^t[1.tr, 1.ref / tr', ref'] \wedge \\ \mathit{disable}(tr, 0.tr, 1.tr) \wedge \\ \mathit{IMTR} \left(\begin{array}{c} tr, tr', 0.tr, 1.tr, \\ ref, ref', 0.ref, 1.ref \end{array} \right) \end{array} \right) ; \mathbf{R1_{ct}}(\mathbf{true}) \wedge \mathbf{true} \\
\vdash \\
((P_f^t ; U0(\mathit{out}\alpha P)) \wedge (\mathit{Skip}_f^t ; U1(\mathit{out}\alpha \mathit{Stop})))_{+\{tr, ref\}} ; (IM1 \vee IM2)
\end{array} \right) \\
& [\mathit{Skip} \text{ and its postcondition}] \\
= \mathbf{R_{ct}} \left(\begin{array}{c}
\neg \left((P_f^f \wedge tr' = tr) ; \mathbf{R1_{ct}}(\mathbf{true}) \right) \wedge \\
\neg \exists \left(\begin{array}{c} 0.tr, 0.ref, \\ 1.tr, 1.ref \end{array} \right) \bullet \left(\begin{array}{c} P_f^f[0.tr, 0.ref / tr', ref'] \wedge \\ (\neg \mathit{wait}' \wedge tr' = tr \wedge \mathit{state}' = \mathit{state})[1.tr, 1.ref / tr', ref'] \wedge \\ \mathit{disable}(tr, 0.tr, 1.tr) \wedge \\ \mathit{IMTR} \left(\begin{array}{c} tr, tr', 0.tr, 1.tr, \\ ref, ref', 0.ref, 1.ref \end{array} \right) \end{array} \right) ; \mathbf{R1_{ct}}(\mathbf{true}) \\
\vdash \\
((P_f^t ; U0(\mathit{out}\alpha P)) \wedge ((\mathit{wait}' \wedge \frown / tr' = \frown / tr) ; U1(\mathit{out}\alpha \mathit{Stop})))_{+\{tr, ref\}} ; (IM1 \vee IM2)
\end{array} \right) \\
& [\text{Lemma 47,48}]
\end{array}
$$

$$
\begin{array}{l}
= \mathbf{R_{ct}} \left(\begin{array}{c}
\neg\, ((P_f^f \wedge tr' = tr)\,;\, \mathbf{R1_{ct}}(\mathbf{true})) \wedge \\
\neg\, \exists \left(\begin{array}{c} 0.tr, 0.ref, \\ 1.tr, 1.ref \end{array} \right) \bullet \left(\begin{array}{c} P_f^f[0.tr, 0.ref/tr', ref'] \wedge \\ (\neg\, wait' \wedge tr' = tr \wedge state' = state)[1.tr, 1.ref/tr', ref'] \wedge \\ tr' = 0.tr \wedge ref' = 0.ref \end{array} \right) ; \mathbf{R1_{ct}}(\mathbf{true}) \\
\vdash \\
((P_f^t\,;\, U0(out\alpha P)) \wedge ((wait' \wedge {}^\frown/tr' = {}^\frown/tr)\,;\, U1(out\alpha Stop)))_{+\{tr,ref\}}\,;\, (IM1 \vee IM2)
\end{array} \right) \\
\qquad\qquad \text{[predicate calculus]} \\
= \mathbf{R_{ct}} \left(\begin{array}{c}
\neg\, ((P_f^f \wedge tr' = tr)\,;\, \mathbf{R1_{ct}}(\mathbf{true})) \wedge \neg\, (P_f^f;\, \mathbf{R1_{ct}}(\mathbf{true})) \\
\vdash \\
((P_f^t\,;\, U0(out\alpha P)) \wedge ((wait' \wedge {}^\frown/tr' = {}^\frown/tr)\,;\, U1(out\alpha Stop)))_{+\{tr,ref\}}\,;\, (IM1 \vee IM2)
\end{array} \right) \\
\qquad\qquad \text{[relational calculus]} \\
= \mathbf{R_{ct}} \left(\begin{array}{c}
\neg\, (P_f^f;\, \mathbf{R1_{ct}}(\mathbf{true})) \\
\vdash \\
((P_f^t\,;\, U0(out\alpha P)) \wedge ((wait' \wedge {}^\frown/tr' = {}^\frown/tr)\,;\, U1(out\alpha Stop)))_{+\{tr,ref\}}\,;\, (IM1 \vee IM2)
\end{array} \right) \\
\qquad\qquad [P \text{ is } \mathbf{CSP4_{ct}}] \\
= \mathbf{R_{ct}} \left(\begin{array}{c}
\neg\, P_f^f \vdash \\
((P_f^t\,;\, U0(out\alpha P)) \wedge ((wait' \wedge {}^\frown/tr' = {}^\frown/tr)\,;\, U1(out\alpha Stop)))_{+\{tr,ref\}}\,;\, (IM1 \vee IM2)
\end{array} \right) \\
\qquad\qquad [P \text{ is } \mathbf{CSP4_{ct}}] \\
= \mathbf{R_{ct}} \left(\begin{array}{c}
\neg\, P_f^f \vdash \\
((P_f^t\,;\, U0(out\alpha P)) \wedge ((wait' \wedge {}^\frown/tr' = {}^\frown/tr)\,;\, U1(out\alpha Stop)))_{+\{tr,ref\}}\,;\, (IM1 \vee IM2)
\end{array} \right) \\
\qquad\qquad \text{[Lemma 47 and } {}^\frown/1.tr = {}^\frown/tr \text{ contradicts with } enable(tr, 0.tr, 1.tr) \text{ in } IM2] \\
= \mathbf{R_{ct}} \left(\begin{array}{c}
\neg\, P_f^f \vdash \\
((P_f^t\,;\, U0(out\alpha P)) \wedge ((wait' \wedge {}^\frown/tr' = {}^\frown/tr)\,;\, U1(out\alpha Stop)))_{+\{tr,ref\}}\,;\, IM1
\end{array} \right) \\
\qquad\qquad \text{[Lemma 48 and relational calculus]} \\
= \mathbf{R_{ct}}(\neg\, P_f^f \vdash P_f^t) \qquad\qquad \text{[Theorem 1]} \\
= P
\end{array}
$$

□

Law 74 *Skip* $\triangle$ $P \sqsubseteq$ *Skip*

Proof. The proof is similar to that of Law 73. □

Law 75 $(P \triangle Chaos) = Chaos = (Chaos \triangle P)$

Proof.

$$
\begin{array}{ll}
& (P \triangle \mathit{Chaos}) \qquad \text{[Definition 18]} \\
= & \mathbf{R_{ct}} \left(\begin{array}{c}
\neg\, (((P_f^f \vee \mathit{Chaos}_f^f) \wedge tr' = tr)\,;\, \mathbf{R1_{ct}}(\mathbf{true})) \wedge \\
\neg\, \exists \left(\begin{array}{c} 0.tr, 0.ref, \\ 1.tr, 1.ref \end{array} \right) \bullet \left(\begin{array}{c} P_f^f[0.tr, 0.ref/tr', ref'] \wedge \\ \mathit{Chaos}_f^t[1.tr, 1.ref/tr', ref'] \wedge \\ \mathit{disable}(tr, 0.tr, 1.tr) \wedge \\ \mathit{IMTR}\left(\begin{array}{c} tr, tr', 0.tr, 1.tr, \\ ref, ref', 0.ref, 1.ref \end{array} \right) \end{array} \right) ;\mathbf{R1_{ct}}(\mathbf{true}) \wedge \\
\neg\, \exists \left(\begin{array}{c} 0.tr, 0.ref, \\ 1.tr, 1.ref \end{array} \right) \bullet \left(\begin{array}{c} P_f^t[0.tr, 0.ref/tr', ref'] \wedge \\ \mathit{Chaos}_f^f[1.tr, 1.ref/tr', ref'] \wedge \\ \mathit{enable}(tr, 0.tr, 1.tr) \wedge \\ \mathit{IMTR}\left(\begin{array}{c} tr, tr', 0.tr, 1.tr, \\ ref, ref', 0.ref, 1.ref \end{array} \right) \end{array} \right) ;\mathbf{R1_{ct}}(\mathbf{true}) \\
\vdash \\
((P_f^t\,;\, U0(\mathit{out}\alpha P)) \wedge (\mathit{Chaos}_f^t\,;\, U1(\mathit{out}\alpha \mathit{Skip})))_{+\{tr,ref\}}\,;\, (\mathit{IM}1 \vee \mathit{IM}2)
\end{array} \right) \\
& \qquad \text{[}\mathit{Chaos}\text{ and its precondition]} \\
= & \mathbf{R_{ct}} \left(\begin{array}{c}
\neg\, (((P_f^f \vee \mathbf{true}) \wedge tr' = tr)\,;\, \mathbf{R1_{ct}}(\mathbf{true})) \wedge \\
\neg\, \exists \left(\begin{array}{c} 0.tr, 0.ref, \\ 1.tr, 1.ref \end{array} \right) \bullet \left(\begin{array}{c} P_f^f[0.tr, 0.ref/tr', ref'] \wedge \\ \mathit{Chaos}_f^t[1.tr, 1.ref/tr', ref'] \wedge \\ \mathit{disable}(tr, 0.tr, 1.tr) \wedge \\ \mathit{IMTR}\left(\begin{array}{c} tr, tr', 0.tr, 1.tr, \\ ref, ref', 0.ref, 1.ref \end{array} \right) \end{array} \right) ;\mathbf{R1_{ct}}(\mathbf{true}) \wedge \\
\neg\, \exists \left(\begin{array}{c} 0.tr, 0.ref, \\ 1.tr, 1.ref \end{array} \right) \bullet \left(\begin{array}{c} P_f^t[0.tr, 0.ref/tr', ref'] \wedge \\ \mathbf{true}[1.tr, 1.ref/tr', ref'] \wedge \\ \mathit{enable}(tr, 0.tr, 1.tr) \wedge \\ \mathit{IMTR}\left(\begin{array}{c} tr, tr', 0.tr, 1.tr, \\ ref, ref', 0.ref, 1.ref \end{array} \right) \end{array} \right) ;\mathbf{R1_{ct}}(\mathbf{true}) \\
\vdash \\
((P_f^t\,;\, U0(\mathit{out}\alpha P)) \wedge (\mathit{Skip}_f^t\,;\, U1(\mathit{out}\alpha \mathit{Stop})))_{+\{tr,ref\}}\,;\, (\mathit{IM}1 \vee \mathit{IM}2)
\end{array} \right) \qquad \text{[rel. calculus]} \\
= & \mathbf{R_{ct}} \left(\begin{array}{c}
\neg\, (\mathbf{true}) \wedge \\
\neg\, \exists \left(\begin{array}{c} 0.tr, 0.ref, \\ 1.tr, 1.ref \end{array} \right) \bullet \left(\begin{array}{c} P_f^f[0.tr, 0.ref/tr', ref'] \wedge \\ \mathit{Chaos}_f^t[1.tr, 1.ref/tr', ref'] \wedge \\ \mathit{disable}(tr, 0.tr, 1.tr) \wedge \\ \mathit{IMTR}\left(\begin{array}{c} tr, tr', 0.tr, 1.tr, \\ ref, ref', 0.ref, 1.ref \end{array} \right) \end{array} \right) ;\mathbf{R1_{ct}}(\mathbf{true}) \wedge \\
\neg\, \exists \left(\begin{array}{c} 0.tr, 0.ref, \\ 1.tr, 1.ref \end{array} \right) \bullet \left(\begin{array}{c} P_f^t[0.tr, 0.ref/tr', ref'] \wedge \\ \mathbf{true}[1.tr, 1.ref/tr', ref'] \wedge \\ \mathit{enable}(tr, 0.tr, 1.tr) \wedge \\ \mathit{IMTR}\left(\begin{array}{c} tr, tr', 0.tr, 1.tr, \\ ref, ref', 0.ref, 1.ref \end{array} \right) \end{array} \right) ;\mathbf{R1_{ct}}(\mathbf{true}) \\
\vdash \\
((P_f^t\,;\, U0(\mathit{out}\alpha P)) \wedge (\mathit{Skip}_f^t\,;\, U1(\mathit{out}\alpha \mathit{Stop})))_{+\{tr,ref\}}\,;\, (\mathit{IM}1 \vee \mathit{IM}2)
\end{array} \right) \qquad \text{[rel. calculus]} \\
= & \mathbf{R_{ct}}(\mathbf{true})
\end{array}
$$

The proof of $\mathit{Chaos} \triangle P = \mathit{Chaps}$ is in a similar manner.

□

E.2 Timed Interrupts

Law 76 $(P \,\Box\, Q) \triangle_d R = (P \triangle_d R) \,\Box\, (Q \triangle_d R)$

Proof. The proof of this law is straightforward but very time-consuming. Also the hand proof is error-prone. Therefore, we leave the proof to the mechanical proof in the future. □

Law 77 $Stop \triangle_d P = Wait\ d\,;P$

Lemma 49. $Stop \triangle_d P = \left(\begin{array}{c} \neg\,((\frown/tr' = \frown/tr \wedge state' = state \wedge \#tr' - \#tr = d);(\neg\, wait \wedge P_f^f)) \\ \vdash \\ (wait' \wedge \frown/tr' = \frown/tr \wedge state' = state \wedge \#tr' - \#tr < d) \vee \\ (\frown/tr' = \frown/tr \wedge state' = state \wedge \#tr' - \#tr = d)\,;(\neg\, wait' \wedge P_f^t) \end{array}\right)$

Proof.

$$
\begin{array}{ll}
& Stop \triangle_d P \qquad \text{[Def-21]} \\
= & \left(\begin{array}{c} \neg\,((Stop_f^f \wedge \#tr' - \#tr \leq d)\,;\mathbf{R1_{ct}}(\mathbf{true})) \wedge \\ \neg\,((Stop_f^t \wedge \#tr' - \#tr = d)\,;(wait \wedge \mathbb{I}^{-wait} \wedge \neg\, wait')\,;P_f^f) \\ \vdash \\ (Stop_f^t \wedge \#tr' - \#tr < d) \vee (Stop_f^t \wedge \#tr' - \#tr \leq d \wedge \neg\, wait') \vee \\ ((Stop_f^t \wedge \#tr' - \#tr = d)\,;(wait \wedge \mathbb{I}^{-wait} \wedge \neg\, wait')\,;P_f^t) \end{array}\right) \qquad \text{[Stop]} \\
= & \left(\begin{array}{c} \neg\,((\mathbf{false} \wedge \#tr' - \#tr \leq d)\,;\mathbf{R1_{ct}}(\mathbf{true})) \wedge \\ \neg\,(((wait' \wedge \frown/tr' = \frown/tr \wedge state' = state) \wedge \#tr' - \#tr = d)\,;(wait \wedge \mathbb{I}^{-wait} \wedge \neg\, wait')\,;P_f^f) \\ \vdash \\ ((wait' \wedge \frown/tr' = \frown/tr \wedge state' = state) \wedge \#tr' - \#tr < d) \vee \\ ((wait' \wedge \frown/tr' = \frown/tr \wedge state' = state) \wedge \#tr' - \#tr \leq d \wedge \neg\, wait') \vee \\ (((wait' \wedge \frown/tr' = \frown/tr \wedge state' = state) \wedge \#tr' - \#tr = d)\,;(wait \wedge \mathbb{I}^{-wait} \wedge \neg\, wait')\,;P_f^t) \end{array}\right) \\
& \qquad \text{[relational calculus]} \\
= & \left(\begin{array}{c} \neg\,(((wait' \wedge \frown/tr' = \frown/tr \wedge state' = state) \wedge \#tr' - \#tr = d)\,;(wait \wedge \mathbb{I}^{-wait} \wedge \neg\, wait')\,;P_f^f) \\ \vdash \\ ((wait' \wedge \frown/tr' = \frown/tr \wedge state' = state) \wedge \#tr' - \#tr < d) \vee \\ (((wait' \wedge \frown/tr' = \frown/tr \wedge state' = state) \wedge \#tr' - \#tr = d)\,;(wait \wedge \mathbb{I}^{-wait} \wedge \neg\, wait')\,;P_f^t) \end{array}\right) \\
& \qquad \text{[relational calculus]} \\
= & \left(\begin{array}{c} \neg\,((\frown/tr' = \frown/tr \wedge state' = state \wedge \#tr' - \#tr = d);(\neg\, wait \wedge P_f^f)) \\ \vdash \\ (wait' \wedge \frown/tr' = \frown/tr \wedge state' = state \wedge \#tr' - \#tr < d) \vee \\ (\frown/tr' = \frown/tr \wedge state' = state \wedge \#tr' - \#tr = d)\,;(\neg\, wait' \wedge P_f^t) \end{array}\right)
\end{array}
$$

Lemma 50. $Wait\ d;P = \mathbf{R_{ct}}\left(\begin{array}{c} \neg\,((\frown/tr' = \frown/tr \wedge \#tr' - \#tr = d)\,;(\neg\, wait \wedge P_f^f)) \\ \vdash \\ (\frown/tr' = \frown/tr \wedge \#tr' - \#tr < d \wedge wait' \wedge state' = state) \vee \\ (\frown/tr' = \frown/tr \wedge \#tr' - \#tr = d \wedge state' = state)\,;(\neg\, wait \wedge P_f^t) \end{array}\right)$

$$
\begin{array}{l}
\mathit{Wait}\ d\,;\, P \qquad\qquad [\text{Theorem 6}] \\
= \mathbf{R_{ct}} \left(\begin{array}{c} \neg\,(\mathbf{R1_{ct}}((\mathit{Wait}\ d)^f_f)\,;\,\mathbf{R1_{ct}}(\mathbf{true})) \wedge \neg\,(\mathbf{R1_{ct}}((\mathit{Wait}\ d)^t_f)\,;\,\mathbf{R1_{ct}}(\neg\ \mathit{wait} \wedge \mathbf{R2_{ct}}(P^f_f))) \\ \vdash \\ \mathbf{R1_{ct}}((\mathit{Wait}\ d)^t_f)\,;\,\mathbf{R1_{ct}}(\mathbb{I} \lhd \mathit{wait} \rhd \mathbf{R2_{ct}}(P^t_f)) \end{array} \right) \\
\qquad\qquad [\mathit{Wait}(\text{Theorem 8})] \\
= \mathbf{R_{ct}} \left(\begin{array}{c} \neg\,(\mathbf{R1_{ct}}(\mathbf{false})\,;\,\mathbf{R1_{ct}}(\mathbf{true})) \wedge \neg\,(\mathbf{R1_{ct}}((\mathit{Wait}\ d)^t_f)\,;\,\mathbf{R1_{ct}}(\neg\ \mathit{wait} \wedge \mathbf{R2_{ct}}(P^f_f))) \\ \vdash \\ \mathbf{R1_{ct}}((\mathit{Wait}\ d)^t_f)\,;\,\mathbf{R1_{ct}}(\mathbb{I} \lhd \mathit{wait} \rhd \mathbf{R2_{ct}}(P^t_f)) \end{array} \right) \\
\qquad\qquad [\text{relational calculus}] \\
= \mathbf{R_{ct}} \left(\begin{array}{c} \neg\,(\mathbf{R1_{ct}}((\mathit{Wait}\ d)^t_f)\,;\,\mathbf{R1_{ct}}(\neg\ \mathit{wait} \wedge \mathbf{R2_{ct}}(P^f_f))) \\ \vdash \\ \mathbf{R1_{ct}}((\mathit{Wait}\ d)^t_f)\,;\,\mathbf{R1_{ct}}(\mathbb{I} \lhd \mathit{wait} \rhd \mathbf{R2_{ct}}(P^t_f)) \end{array} \right) \\
\qquad\qquad [\mathit{Wait}(\text{Theorem 8}) \text{ and relational calculus}] \\
= \mathbf{R_{ct}} \left(\begin{array}{c} \neg\,(\mathbf{R1_{ct}}(\frown/tr' = \frown/tr \wedge \#tr' - \#tr < d \wedge \mathit{wait}')\,;\,\mathbf{R1_{ct}}(\neg\ \mathit{wait} \wedge \mathbf{R2_{ct}}(P^f_f))) \vee \\ \neg\,(\mathbf{R1_{ct}}(\frown/tr' = \frown/tr \wedge \#tr' - \#tr = d \wedge \neg\ \mathit{wait}' \wedge \mathit{state}' = \mathit{state})\,;\,\mathbf{R1_{ct}}(\neg\ \mathit{wait} \wedge \mathbf{R2_{ct}}(P^f_f))) \\ \vdash \\ \mathbf{R1_{ct}}(\frown/tr' = \frown/tr \wedge \#tr' - \#tr < d \wedge \mathit{wait}' \wedge \mathit{state}' = \mathit{state})\,;\,\mathbf{R1_{ct}}(\mathbb{I} \lhd \mathit{wait} \rhd \mathbf{R2_{ct}}(P^t_f)) \vee \\ \mathbf{R1_{ct}}(\frown/tr' = \frown/tr \wedge \#tr' - \#tr = d \wedge \neg\ \mathit{wait}' \wedge \mathit{state}' = \mathit{state})\,;\,\mathbf{R1_{ct}}(\mathbb{I} \lhd \mathit{wait} \rhd \mathbf{R2_{ct}}(P^t_f)) \end{array} \right) \\
\qquad\qquad [\text{relational calculus}] \\
= \mathbf{R_{ct}} \left(\begin{array}{c} \neg\,(\mathbf{false}) \vee \\ \neg\,(\mathbf{R1_{ct}}(\frown/tr' = \frown/tr \wedge \#tr' - \#tr = d \wedge \neg\ \mathit{wait}' \wedge \mathit{state}' = \mathit{state})\,;\,\mathbf{R1_{ct}}(\neg\ \mathit{wait} \wedge \mathbf{R2_{ct}}(P^f_f))) \\ \vdash \\ \mathbf{R1_{ct}}(\frown/tr' = \frown/tr \wedge \#tr' - \#tr < d \wedge \mathit{wait}' \wedge \mathit{state}' = \mathit{state})\,;\,\mathbf{R1_{ct}}(\mathbb{I} \wedge \mathit{wait}) \vee \\ \mathbf{R1_{ct}}(\frown/tr' = \frown/tr \wedge \#tr' - \#tr = d \wedge \neg\ \mathit{wait}' \wedge \mathit{state}' = \mathit{state})\,;\,\mathbf{R1_{ct}}(\neg\ \mathit{wait} \wedge \mathbf{R2_{ct}}(P^t_f)) \end{array} \right) \\
\qquad\qquad [P \text{ is } \mathbf{R1_{ct}}, \mathbf{R2_{ct}} \text{ and relational calculus}] \\
= \mathbf{R_{ct}} \left(\begin{array}{c} \neg\,(\frown/tr' = \frown/tr \wedge \#tr' - \#tr = d)\,;\,(\neg\ \mathit{wait} \wedge P^f_f)) \\ \vdash \\ \mathbf{R1_{ct}}(\frown/tr' = \frown/tr \wedge \#tr' - \#tr < d \wedge \mathit{wait}' \wedge \mathit{state}' = \mathit{state})\,;\,\mathbf{R1_{ct}}(\mathbb{I} \wedge \mathit{wait}) \vee \\ \mathbf{R1_{ct}}(\frown/tr' = \frown/tr \wedge \#tr' - \#tr = d \wedge \neg\ \mathit{wait}' \wedge \mathit{state}' = \mathit{state})\,;\,(\neg\ \mathit{wait} \wedge P^t_f) \end{array} \right) \\
\qquad\qquad [\text{relational calculus}] \\
= \mathbf{R_{ct}} \left(\begin{array}{c} \neg\,((\frown/tr' = \frown/tr \wedge \#tr' - \#tr = d)\,;\,(\neg\ \mathit{wait} \wedge P^f_f)) \\ \vdash \\ (\frown/tr' = \frown/tr \wedge \#tr' - \#tr < d \wedge \mathit{wait}' \wedge \mathit{state}' = \mathit{state}) \vee \\ (\frown/tr' = \frown/tr \wedge \#tr' - \#tr = d \wedge \mathit{state}' = \mathit{state})\,;\,(\neg\ \mathit{wait} \wedge P^t_f) \end{array} \right)
\end{array}
$$

□

Law 78 $\mathit{Skip} \triangle_d P = \mathit{Skip}$ *if* $d > 0$

Proof.

$$
\begin{array}{ll}
& \mathit{Skip} \triangle_d P & [\text{Def-21}] \\
= & \left(\begin{array}{c} \neg\,((\mathit{Skip}_f^f \wedge \#tr' - \#tr \leq d)\,;\, \mathbf{R1_{ct}}(\mathbf{true})) \wedge \\ \neg\,((\mathit{Skip}_f^t \wedge \#tr' - \#tr = d)\,;\, (\mathit{wait} \wedge \mathbb{I}^{-\mathit{wait}} \wedge \neg\, \mathit{wait}')\,;\, P_f^f) \\ \vdash \\ (\mathit{Skip}_f^t \wedge \#tr' - \#tr < d) \vee (\mathit{Skip}_f^t \wedge \#tr' - \#tr \leq d \wedge \neg\, \mathit{wait}') \vee \\ ((\mathit{Skip}_f^t \wedge \#tr' - \#tr = d)\,;\, (\mathit{wait} \wedge \mathbb{I}^{-\mathit{wait}} \wedge \neg\, \mathit{wait}')\,;\, P_f^t) \end{array} \right) & [\mathit{Skip}] \\
= & \left(\begin{array}{c} \neg\,((\mathbf{false} \wedge \#tr' - \#tr \leq d)\,;\, \mathbf{R1_{ct}}(\mathbf{true})) \wedge \\ \neg\,(((\neg\, \mathit{wait}' \wedge tr' = tr \wedge \mathit{state}' = \mathit{state}) \wedge \#tr' - \#tr = d)\,;\, (\mathit{wait} \wedge \mathbb{I}^{-\mathit{wait}} \wedge \neg\, \mathit{wait}')\,;\, P_f^f) \\ \vdash \\ ((\neg\, \mathit{wait}' \wedge tr' = tr \wedge \mathit{state}' = \mathit{state}) \wedge \#tr' - \#tr < d) \vee \\ ((\neg\, \mathit{wait}' \wedge tr' = tr \wedge \mathit{state}' = \mathit{state}) \wedge \#tr' - \#tr \leq d \wedge \neg\, \mathit{wait}') \vee \\ (((\neg\, \mathit{wait}' \wedge tr' = tr \wedge \mathit{state}' = \mathit{state}) \wedge \#tr' - \#tr = d)\,;\, (\mathit{wait} \wedge \mathbb{I}^{-\mathit{wait}} \wedge \neg\, \mathit{wait}')\,;\, P_f^t) \end{array} \right) \\
& & [\text{relational calculus}] \\
= & \left(\begin{array}{c} \mathbf{true} \vdash \\ ((\neg\, \mathit{wait}' \wedge tr' = tr \wedge \mathit{state}' = \mathit{state}) \wedge \#tr' - \#tr < d) \vee \\ ((\neg\, \mathit{wait}' \wedge tr' = tr \wedge \mathit{state}' = \mathit{state}) \wedge \#tr' - \#tr \leq d \wedge \neg\, \mathit{wait}') \end{array} \right) & [\text{property of } \#] \\
= & \left(\begin{array}{c} \mathbf{true} \vdash \\ ((\neg\, \mathit{wait}' \wedge tr' = tr \wedge \mathit{state}' = \mathit{state}) \wedge 0 < d) \vee \\ ((\neg\, \mathit{wait}' \wedge tr' = tr \wedge \mathit{state}' = \mathit{state}) \wedge 0 \leq d \wedge \neg\, \mathit{wait}') \end{array} \right) & [\text{assumption } d > 0] \\
= & \left(\mathbf{true} \vdash \neg\, \mathit{wait}' \wedge tr' = tr \wedge \mathit{state}' = \mathit{state} \right) & [\mathit{Skip}] \\
= & \mathit{Skip} &
\end{array}
$$

□

Law 79 $(\mathit{Wait}\ d\,;\, P) \triangle_{d+d'} Q = \mathit{Wait}\ d\,;\, (P \triangle_{d'} Q)$ *if* $d' > 0$

Lemma 51. $(\mathit{Wait}\ d\,;\, P) \triangle_{d+d'} Q$

Proof.

$$
\begin{array}{ll}
& (\mathit{Wait}\ d\,;\, P) \triangle_{d+d'} Q \qquad \text{[Def-21]} \\
= & \left(\begin{array}{c} \neg\,(((\mathit{Wait}\ d\,;\, P)^f_f \wedge \#tr' - \#tr \leq d)\,;\, \mathbf{R1_{ct}}(\mathbf{true})) \wedge \\ \neg\,(((\mathit{Wait}\ d\,;\, P)^t_f \wedge \#tr' - \#tr = d + d')\,;\, (\mathit{wait} \wedge \mathit{II}^{-\mathit{wait}} \wedge \neg\ \mathit{wait}')\,;\, Q^f_f) \\ \vdash \\ ((\mathit{Wait}\ d\,;\, P)^t_f \wedge \#tr' - \#tr < d + d' \wedge \mathit{wait}') \vee \\ ((\mathit{Wait}\ d\,;\, P)^t_f \wedge \#tr' - \#tr \leq d + d' \wedge \neg\ \mathit{wait}') \vee \\ (((\mathit{Wait}\ d\,;\, P)^t_f \wedge \#tr' - \#tr = d + d')\,;\, (\mathit{wait} \wedge \mathit{II}^{-\mathit{wait}} \wedge \neg\ \mathit{wait}')\,;\, Q^t_f) \end{array}\right) \\
& \qquad \text{[Lemma 50 and its precondition]} \\
= & \left(\begin{array}{c} \neg\,((((\frown/tr' = \frown/tr \wedge \#tr' - \#tr = d)\,;\, (\neg\ \mathit{wait} \wedge P^f_f)) \wedge \#tr' - \#tr \leq d + d')\,;\, \mathbf{R1_{ct}}(\mathbf{true})) \wedge \\ \neg\,(((\mathit{Wait}\ d\,;\, P)^t_f \wedge \#tr' - \#tr = d + d')\,;\, (\mathit{wait} \wedge \mathit{II}^{-\mathit{wait}} \wedge \neg\ \mathit{wait}')\,;\, Q^f_f) \\ \vdash \\ ((\mathit{Wait}\ d\,;\, P)^t_f \wedge \#tr' - \#tr < d + d' \wedge \mathit{wait}') \vee \\ ((\mathit{Wait}\ d\,;\, P)^t_f \wedge \#tr' - \#tr \leq d + d' \wedge \neg\ \mathit{wait}') \vee \\ (((\mathit{Wait}\ d\,;\, P)^t_f \wedge \#tr' - \#tr = d + d')\,;\, (\mathit{wait} \wedge \mathit{II}^{-\mathit{wait}} \wedge \neg\ \mathit{wait}')\,;\, Q^t_f) \end{array}\right) \\
& \qquad \text{[relational calculus and property of \#]} \\
= & \left(\begin{array}{c} \neg\,((\frown/tr' = \frown/tr \wedge \#tr' - \#tr = d)\,;\, (\neg\ \mathit{wait} \wedge P^f_f \wedge \#tr' - \#tr \leq d')\,;\, \mathbf{R1_{ct}}(\mathbf{true})) \wedge \\ \neg\left(\left(\left(\left(\begin{array}{c}(\frown/tr' = \frown/tr \wedge \#tr' - \#tr < d \wedge \mathit{wait}') \\ \vee \left(\begin{array}{c}\frown/tr' = \frown/tr \wedge \\ \#tr' - \#tr = d \wedge \\ \mathit{state}' = \mathit{state}\end{array}\right)\,;\, (\neg\ \mathit{wait} \wedge P^t_f)\end{array}\right) \wedge \#tr' - \#tr = d + d'\right)\,;\, (\mathit{wait} \wedge \mathit{II}^{-\mathit{wait}} \wedge \neg\ \mathit{wait}')\,;\, Q^f_f\right)\right) \\ \vdash \\ \left(\left(\left(\begin{array}{c}(\frown/tr' = \frown/tr \wedge \#tr' - \#tr < d \wedge \mathit{wait}' \wedge \mathit{state}' = \mathit{state}) \\ \vee \left(\begin{array}{c}\frown/tr' = \frown/tr \wedge \\ \#tr' - \#tr = d \wedge \\ \mathit{state}' = \mathit{state}\end{array}\right)\,;\, (\neg\ \mathit{wait} \wedge P^t_f)\end{array}\right) \wedge \#tr' - \#tr < d + d' \wedge \mathit{wait}'\right) \vee \right. \\ \left(\left(\left(\begin{array}{c}(\frown/tr' = \frown/tr \wedge \#tr' - \#tr < d \wedge \mathit{wait}' \wedge \mathit{state}' = \mathit{state}) \\ \vee \left(\begin{array}{c}\frown/tr' = \frown/tr \wedge \\ \#tr' - \#tr = d \wedge \\ \mathit{state}' = \mathit{state}\end{array}\right)\,;\, (\neg\ \mathit{wait} \wedge P^t_f)\end{array}\right) \wedge \#tr' - \#tr \leq d + d' \wedge \neg\ \mathit{wait}'\right) \vee \right. \\ \left.\left(\left(\left(\left(\begin{array}{c}\left(\begin{array}{c}\frown/tr' = \frown/tr \wedge \#tr' - \#tr < d \\ \wedge\ \mathit{wait}' \wedge \mathit{state}' = \mathit{state}\end{array}\right) \\ \vee \left(\begin{array}{c}\frown/tr' = \frown/tr \wedge \\ \#tr' - \#tr = d \wedge \\ \mathit{state}' = \mathit{state}\end{array}\right)\,;\, (\neg\ \mathit{wait} \wedge P^t_f)\end{array}\right) \wedge \#tr' - \#tr = d + d'\right)\,;\, (\mathit{wait} \wedge \mathit{II}^{-\mathit{wait}} \wedge \neg\ \mathit{wait}')\,;\, Q^t_f\right)\right) \end{array}\right) \\
& \qquad \text{[relational calculus]} \\
= & \left(\begin{array}{c} \neg\,((\frown/tr' = \frown/tr \wedge \#tr' - \#tr = d)\,;\, (\neg\ \mathit{wait} \wedge P^f_f \wedge \#tr' - \#tr \leq d')\,;\, \mathbf{R1_{ct}}(\mathbf{true})) \wedge \\ \neg\left((\frown/tr' = \frown/tr \wedge \#tr' - \#tr = d)\,;\, \left(\begin{array}{l}(\neg\ \mathit{wait} \wedge P^t_f \wedge \#tr' - \#tr = d')\,; \\ (\mathit{wait} \wedge \mathit{II}^{-\mathit{wait}} \wedge \neg\ \mathit{wait}')\,;\, Q^f_f\end{array}\right)\right) \\ \vdash \\ (\frown/tr' = \frown/tr \wedge \#tr' - \#tr < d \wedge \mathit{wait}' \wedge \mathit{state}' = \mathit{state}) \vee \\ ((\frown/tr' = \frown/tr \wedge \#tr' - \#tr = d \wedge \neg\ \mathit{wait}' \wedge \mathit{state}' = \mathit{state})\,;\, (\neg\ \mathit{wait} \wedge P^t_f \wedge \mathit{wait}' \wedge \#tr' - \#tr < d')) \vee \\ ((\frown/tr' = \frown/tr \wedge \#tr' - \#tr = d \wedge \neg\ \mathit{wait}' \wedge \mathit{state}' = \mathit{state})\,;\, (\neg\ \mathit{wait} \wedge P^t_f \wedge \mathit{wait}' \wedge \#tr' - \#tr \leq d')) \vee \\ \left(\left(\begin{array}{c}\frown/tr' = \frown/tr \wedge \#tr' - \#tr = d \\ \wedge\ \mathit{state}' = \mathit{state}\end{array}\right)\,;\, (\neg\ \mathit{wait} \wedge P^t_f \wedge \#tr' - \#tr \leq d')\,;\, (\mathit{wait} \wedge \mathit{II}^{-\mathit{wait}} \wedge \neg\ \mathit{wait}')\,;\, Q^t_f\right) \end{array}\right) \\
& \qquad \text{[relational calculus]}
\end{array}
$$

$$= \left(\begin{array}{c} \neg\left((\frown/tr' = \frown/tr \wedge \#tr' - \#tr = d)\,;(\neg\, wait \wedge P_f^f \wedge \#tr' - \#tr \leq d')\,;\mathbf{R1_{ct}}(\mathbf{true})\right) \wedge \\ \neg\left((\frown/tr' = \frown/tr \wedge \#tr' - \#tr = d)\,;\left(\begin{array}{l}(\neg\, wait \wedge P_f^t \wedge \#tr' - \#tr = d')\,; \\ (wait \wedge \mathbb{I}^{-wait} \wedge \neg\, wait')\,;\, Q_f^f\end{array}\right)\right) \\ \vdash \\ (\frown/tr' = \frown/tr \wedge \#tr' - \#tr < d \wedge wait' \wedge state' = state) \vee \\ ((\frown/tr' = \frown/tr \wedge \#tr' - \#tr = d \wedge state' = state);(\neg\, wait \wedge P_f^t \wedge wait' \wedge \#tr' - \#tr \leq d')) \vee \\ \left(\left(\begin{array}{c}\frown/tr' = \frown/tr \wedge \#tr' - \#tr = d \\ \wedge\, state' = state\end{array}\right);(\neg\, wait \wedge P_f^t \wedge \#tr' - \#tr \leq d');(wait \wedge \mathbb{I}^{-wait} \wedge \neg\, wait')\,;\, Q_f^t\right) \end{array} \right)$$

□

Lemma 52. *Wait* $d\,;(P \triangle_{d'} Q)$

Proof.

$$\begin{array}{ll} & \textit{Wait}\ d\,;(P \triangle_{d'} Q) \qquad \text{[sequential composition]} \\ = & \mathbf{R_{ct}}\left(\begin{array}{c}\neg\,(\mathbf{R1_{ct}}((\textit{Wait}\ d)_f^f)\,;\mathbf{R1_{ct}}(\mathbf{true})) \wedge \\ \neg\,(\mathbf{R1_{ct}}((\textit{Wait}\ d)_f^t)\,;\mathbf{R1_{ct}}(\neg\, wait \wedge \mathbf{R2_{ct}}((P \triangle_{d'} Q)_f^f))) \\ \vdash \\ \mathbf{R1_{ct}}((\textit{Wait}\ d)_f^t)\,;\mathbf{R1_{ct}}(\mathbb{I} \lhd wait \rhd \mathbf{R2_{ct}}((P \triangle_{d'} Q)_f^t))\end{array}\right) \qquad \text{[}\textit{Wait}\text{ and its precondition]} \\ = & \mathbf{R_{ct}}\left(\begin{array}{c}\neg\,(\mathbf{R1_{ct}}(\mathbf{false})\,;\mathbf{R1_{ct}}(\mathbf{true})) \wedge \\ \neg\,(\mathbf{R1_{ct}}((\textit{Wait}\ d)_f^t)\,;\mathbf{R1_{ct}}(\neg\, wait \wedge \mathbf{R2_{ct}}((P \triangle_{d'} Q)_f^f))) \\ \vdash \\ \mathbf{R1_{ct}}((\textit{Wait}\ d)_f^t)\,;\mathbf{R1_{ct}}(\mathbb{I} \lhd wait \rhd \mathbf{R2_{ct}}((P \triangle_{d'} Q)_f^t))\end{array}\right) \qquad \text{[relational calculus]} \\ = & \mathbf{R_{ct}}\left(\begin{array}{c}\neg\,(\mathbf{R1_{ct}}((\textit{Wait}\ d)_f^t)\,;\mathbf{R1_{ct}}(\neg\, wait \wedge \mathbf{R2_{ct}}((P \triangle_{d'} Q)_f^f))) \\ \vdash \\ \mathbf{R1_{ct}}((\textit{Wait}\ d)_f^t)\,;\mathbf{R1_{ct}}(\mathbb{I} \lhd wait \rhd \mathbf{R2_{ct}}((P \triangle_{d'} Q)_f^t))\end{array}\right) \\ & \qquad \text{[}\textit{Wait}\text{ and }\triangle_{d'}\text{ are }\mathbf{R1_{ct}}\text{ and }\mathbf{R2_{ct}}\text{]} \\ = & \mathbf{R_{ct}}\left(\begin{array}{c}\neg\,((\textit{Wait}\ d)_f^t\,;(\neg\, wait \wedge (P \triangle_{d'} Q)_f^f)) \\ \vdash \\ (\textit{Wait}\ d)_f^t\,;(\mathbb{I} \lhd wait \rhd (P \triangle_{d'} Q)_f^t)\end{array}\right) \qquad \text{[}\textit{Wait}\text{ and its precondition and rel. calculus]} \\ = & \mathbf{R_{ct}}\left(\begin{array}{c}\neg\,((\frown/tr' = \frown/tr \wedge state' = state \wedge \#tr' - \#tr = d)\,;(\neg\, wait \wedge (P \triangle_{d'} Q)_f^f)) \\ \vdash \\ ((\frown/tr' = \frown/tr \wedge state' = state \wedge \#tr' - \#tr < d);\ (\mathbb{I} \wedge wait)) \vee \\ ((\frown/tr' = \frown/tr \wedge state' = state \wedge \#tr' - \#tr = d)\,;(\neg\, wait \wedge (P \triangle_{d'} Q)_f^t))\end{array}\right) \\ & \qquad \text{[relational calculus]} \\ = & \mathbf{R_{ct}}\left(\begin{array}{c}\neg\,((\frown/tr' = \frown/tr \wedge state' = state \wedge \#tr' - \#tr = d)\,;(\neg\, wait \wedge (P \triangle_{d'} Q)_f^f)) \\ \vdash \\ ((\frown/tr' = \frown/tr \wedge state' = state \wedge \#tr' - \#tr < d \wedge wait') \vee \\ ((\frown/tr' = \frown/tr \wedge state' = state \wedge \#tr' - \#tr = d)\,;(\neg\, wait \wedge (P \triangle_{d'} Q)_f^t))\end{array}\right) \\ & \qquad \text{[Timed interrupt(Definition 21) and its precondition]} \end{array}$$

$$
=\mathbf{R_{ct}}\left(\begin{array}{c}
\neg\left(\left(\begin{array}{c}\frown/tr' = \frown/tr \wedge \\ state' = state \wedge \\ \#tr' - \#tr = d\end{array}\right); \left(\neg\ wait \wedge \left(\begin{array}{c}((P_f^f \wedge \#tr' - \#tr \leq d'); \mathbf{R1_{ct}}(\mathbf{true})) \\ \vee \left(\begin{array}{c}(P_f^t \wedge \#tr' - \#tr = d'); \\ (wait \wedge \mathbb{I}^{-wait} \wedge \neg\ wait'); Q_f^f\end{array}\right)\end{array}\right)\right)\right) \\
\vdash \\
((\frown/tr' = \frown/tr \wedge state' = state \wedge \#tr' - \#tr < d \wedge wait') \vee \\
((\frown/tr' = \frown/tr \wedge state' = state \wedge \#tr' - \#tr = d); (\neg\ wait \wedge (P \triangle_{d'} Q)_f^t))
\end{array}\right)
$$

[relational calculus]

$$
=\mathbf{R_{ct}}\left(\begin{array}{c}
\neg((\frown/tr' = \frown/tr \wedge \#tr' - \#tr = d); (\neg\ wait \wedge P_f^f \wedge \#tr' - \#tr \leq d'); \mathbf{R1_{ct}}(\mathbf{true})) \wedge \\
\neg\left((\frown/tr' = \frown/tr \wedge \#tr' - \#tr = d); \left(\begin{array}{c}(\neg\ wait \wedge P_f^t \wedge \#tr' - \#tr = d'); \\ (wait \wedge \mathbb{I}^{-wait} \wedge \neg\ wait'); Q_f^f\end{array}\right)\right) \\
\vdash \\
((\frown/tr' = \frown/tr \wedge state' = state \wedge \#tr' - \#tr < d \wedge wait') \vee \\
((\frown/tr' = \frown/tr \wedge state' = state \wedge \#tr' - \#tr = d); (\neg\ wait \wedge (P \triangle_{d'} Q)_f^t))
\end{array}\right)
$$

[Timed interrupt(Definition 21) and its postcondition]

$$
=\mathbf{R_{ct}}\left(\begin{array}{c}
\neg((\frown/tr' = \frown/tr \wedge \#tr' - \#tr = d); (\neg\ wait \wedge P_f^f \wedge \#tr' - \#tr \leq d'); \mathbf{R1_{ct}}(\mathbf{true})) \wedge \\
\neg\left((\frown/tr' = \frown/tr \wedge \#tr' - \#tr = d); \left(\begin{array}{c}(\neg\ wait \wedge P_f^t \wedge \#tr' - \#tr = d'); \\ (wait \wedge \mathbb{I}^{-wait} \wedge \neg\ wait'); Q_f^f\end{array}\right)\right) \\
\vdash \\
((\frown/tr' = \frown/tr \wedge state' = state \wedge \#tr' - \#tr < d \wedge wait') \vee \\
\left(\left(\begin{array}{c}\frown/tr' = \frown/tr \wedge \\ state' = state \wedge \\ \#tr' - \#tr = d\end{array}\right); \left(\neg\ wait \wedge \left(\begin{array}{c}(P_f^t \wedge \#tr' - \#tr < d' \wedge wait') \vee \\ (P_f^t \wedge \#tr' - \#tr \leq d' \wedge \neg\ wait') \vee \\ ((P_f^t \wedge \#tr' - \#tr = d'); (wait \wedge \mathbb{I}^{-wait} \wedge \neg\ wait'); Q_f^t)\end{array}\right)\right)\right)
\end{array}\right)
$$

[Timed interrupt(Definition 21) and its postcondition]

$$
=\mathbf{R_{ct}}\left(\begin{array}{c}
\neg((\frown/tr' = \frown/tr \wedge \#tr' - \#tr = d); (\neg\ wait \wedge P_f^f \wedge \#tr' - \#tr \leq d'); \mathbf{R1_{ct}}(\mathbf{true})) \wedge \\
\neg\left((\frown/tr' = \frown/tr \wedge \#tr' - \#tr = d); \left(\begin{array}{c}(\neg\ wait \wedge P_f^t \wedge \#tr' - \#tr = d'); \\ (wait \wedge \mathbb{I}^{-wait} \wedge \neg\ wait'); Q_f^f\end{array}\right)\right) \\
\vdash \\
((\frown/tr' = \frown/tr \wedge state' = state \wedge \#tr' - \#tr < d \wedge wait') \vee \\
((\frown/tr' = \frown/tr \wedge state' = state \wedge \#tr' - \#tr = d); (\neg\ wait \wedge P_f^t \wedge \#tr' - \#tr < d' \wedge wait')) \vee \\
((\frown/tr' = \frown/tr \wedge state' = state \wedge \#tr' - \#tr = d); (\neg\ wait \wedge P_f^t \wedge \#tr' - \#tr \leq d' \wedge \neg\ wait')) \vee \\
\left(\left(\begin{array}{c}\frown/tr' = \frown/tr \wedge state' = state \\ \wedge\ \#tr' - \#tr = d\end{array}\right); (\neg\ wait \wedge P_f^t \wedge \#tr' - \#tr = d'); (wait \wedge \mathbb{I}^{-wait} \wedge \neg\ wait'); Q_f^t\right)
\end{array}\right)
$$

□

Law 80 $(Wait\ (d + d'); P) \triangle_d Q = Wait\ d; Q$

Proof. The proof is very similar to that of Law 79. □

References

[1] A. W. Roscoe. Model-checking CSP. In *A Classical Mind: essays in Honour of C.A.R. Hoare*, chapter 21. Prentice-Hall, 1994.

[2] A. Cavalcanti and J. Woodcock. Zrc - a refinement calculus for z. *Formal Asp. Comput.*, 10(3):267–289, 1998.

[3] A. Cavalcanti and J. Woodcock. A tutorial introduction to CSP in Unifying Theories of Programming. In *Refinement Techniques in Software Engineering*, volume 3167 of *LNCS*, pages 220–268. Springer, 2006.

[4] A. L. C. Cavalcanti, A. C. A. Sampaio, and J. C. P. Woodcock. A Refinement Strategy for *Circus*. *Formal Aspects of Computing*, 15(2 - 3):146 — 181, 2003.

[5] A. L. C. Cavalcanti, A. C. A. Sampaio, and J. C. P. Woodcock. Unifying Classes and Processes. *Software and System Modelling*, 4(3):277 – 296, 2005.

[6] C. Fidge, I. Hayes, and G. Watson. The Deadline Command. *IEE Proceedings—Software*, 146:104–111, 1998.

[7] I. J. Hayes and M. Utting. A sequential real-time refinement calculus. *Acta Inf.*, pages 385–448, 2001.

[8] C. A. R. Hoare. An axiomatic basis for computer programming. *Commun. ACM*, 12(10):576–580, Oct. 1969.

[9] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall International, 1985.

[10] C. A. R. Hoare and H. Jifeng. *Unifying Theories of Programming*. Prentice-Hall International, 1998.

[11] A. McEwan and J. Woodcock. Unifying theories of interrupts. In *proceedings of the second UTP Symposium, Trinity College Dublin*, 2008.

[12] C. Morgan. *Programming from specifications*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1990.

[13] M. Oliveira, A. Cavalcanti, and J. Woodcock. A UTP Semantics for *Circus*. *Formal Aspects of Computing*, 21(1):3 – 32, 2007.

[14] M. Oliveira, A. Cavalcanti, and J. Woodcock. Unifying theories in ProofPower-Z. *Formal Aspects of Computing Journal*, 2007.

[15] S. Peuker and I. Hayes. Reasoning about deadlines in concurrent real-time programs. In *IPDPS '03: Proceedings of the 17th International Symposium on Parallel and Distributed Processing*, page 237.1, Washington, DC, USA, 2003. IEEE Computer Society.

[16] S. Qin, J. S. Dong, and W.-N. Chin. A semantic foundation for TCOZ in unifying theories of programming. In *FME'03*, pages 321–340, 2003.

[17] A. W. Roscoe. *The Theory and Practice of Concurrency*. Prentice-Hall International, 1998.

[18] S. A. Schneider. *Concurrent and real-time systems: the CSP approach*. John Wiley & Sons, 1999.

[19] D. Scholefield, H. Zedan, and H. Jifeng. A specification-oriented semantics for the refinement of real-time systems. *Theor. Comput. Sci.*, 131:219–241, August 1994.

[20] A. Sherif, A. L. C. Cavalcanti, H. Jifeng, and A. C. A. Sampaio. A process algebraic framework for specification and validation of real-time systems. *Formal Aspects of Computing*, 22(2):153 – 191, 2010.

[21] A. Sherif and J. He. Towards a time model for *Circus*. In *ICFEM '02: Proceedings of the 4th International Conference on Formal Engineering Methods*, pages 613–624, London, UK, 2002. Springer-Verlag.

[22] K. Wei, J. Woodcock, and A. Burns. A timed model of *Circus* with the reactive design miracle. In *8th International Conference on Software Engineering and Formal Methods (SEFM)*, pages 315–319, Pisa, Italy, September 2010. IEEE Computer Society.

[23] K. Wei, J. Woodcock, and A. Cavalcanti. New *Circus Time*. Technical report, Department of Computer Science, University of York, UK, March 2012. Avaliable at http://www.cs.york.ac.uk/circus/hijac/publication.html.

[24] J. Woodcock. The miracle of reactive programming. In *Unifying Theories of Programming 2008: 2nd International Symposium*, Dublin, Ireland, 2008. Springer-Verlag.

[25] J. Woodcock and J. Davies. *Using Z: Specification, Refinement and Proof*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1996.

[26] J. C. P. Woodcock and A. L. C. Cavalcanti. A Tutorial Introduction to Designs in Unifying Theories of Programming. In *IFM 2004*, volume 2999 of *LNCS*, pages 40 – 66.

[27] J. C. P. Woodcock and A. L. C. Cavalcanti. A concurrent language for refinement. In A. Butterfield and C. Pahl, editors, *IWFM'01: 5th Irish Workshop in Formal Methods*, BCS Electronic Workshops in Computing, Dublin, Ireland, July 2001.

[28] J. C. P. Woodcock and A. L. C. Cavalcanti. The semantics of *circus*. In D. Bert, J. P. Bowen, M. C. Henson, and K. Robinson, editors, *ZB 2002: Formal Specification and Development in Z and B*, volume 2272 of *Lecture Notes in Computer Science*, pages 184—203. Springer-Verlag, 2002.

[29] F. Zeyda and A. Cavalcanti. Mechanical reasoning about families of UTP theories. In *Electronic Notes in Theoretical Computer Science*, pages 240:239–257, July 2009.