Specification coverage for testing in *Circus*

Ana Cavalcanti[1] and Marie-Claude Gaudel[2]

[1] University of York, Department of Computer Science
York YO10 5DD, UK
[2] LRI, Université de Paris-Sud and CNRS
Orsay 91405, France

Abstract. The Unifying Theories of Programming underpins the development of *Circus*, a state-rich process algebra for refinement. We have previously presented a theory of testing for *Circus*; it gives a symbolic characterisation of tests. Each symbolic test specifies constraints that capture the effect of the possibly nondeterministic state operations, and their interaction. This is a sound basis for testing techniques based on constraint solving for generation of concrete tests, but does not support well selection criteria based on the structure of the specification. We present here a labelled transition system that captures information about a *Circus* model and its structure. It is useful for testing techniques based on specification coverage. The soundness argument for the new transition system follows the UTP style, but relates the new transition relation to the *Circus* relational model and its operational semantics.

1 Introduction

We have recently proposed a theory of testing for *Circus* [24], a state-rich process algebra that combines Z [37], CSP [28], and a refinement calculus [22]. Its semantics is based on the Unifying Theories of Programming (UTP) [18]. Tutorial introductions to the UTP can be found in [35, 9].

Formal specifications have been widely applied as a starting point for software testing; a few approaches can be found in [10, 14, 2, 3, 1, 20]. Our testing theory for *Circus* [6] instantiates Gaudel's long-standing theory of formal testing [15]. Its foundation is the *Circus* operational semantics [36], which is described and justified in the context of the UTP theory for *Circus* [24].

The main distinguishing feature of the *Circus* testing theory is its symbolic nature: it provides a symbolic characterisation of traces, acceptances and initials, and, most importantly, tests and exhaustive test sets. This takes advantage of the symbolic nature of the *Circus* operational semantics, where unknown data values, such as an input or the result of a nondeterministic choice, are represented by loose constants, which we call symbolic variables. Tractability stems first from the use of alphabets (of symbolic variables) in a manner akin to the use of alphabets (of observational variables) in the UTP. Additionally, we exploit a characterisation of process states as predicates of the UTP theory of relations, using the light touch of the UTP approach for clarity and simplicity.

The conformance relation considered in the *Circus* testing theory is process refinement, which is characterised using the UTP notion of refinement. As usual in testing, we consider divergence-free processes. We take the typical view that, in a specification (as opposed to the system under test) divergence is a mistake and should not be used. In other words, in the specification, divergence should not be indicated as an allowed behaviour. Furthermore, in a program (as opposed to a program model) divergence cannot be distinguished from deadlock.

In previous work, we have shown that, for divergence-free processes, refinement can be characterised by the conjunction of traces refinement and $conf$. This is justified in the UTP in [7], based on a relationship between the UTP and the failures-divergences models of CSP. The *conf* relation [3] has been widely explored in the testing community, and requires reduction of deadlock.

The (symbolic) exhaustive test sets for both traces refinement and *conf* are potentially infinite. Practical techniques rely on selection criteria both to generate and to select a finite set of tests. Together, exhaustiveness and the selection criteria justify the conclusions that can be reached from testing experiments.

The symbolic tests and test sets of *Circus* are ideal as a starting point to consider well-known selection criteria based on constraints decomposition and solving [1, 11, 17]. These allow us to explore the rich data models and ensure meaningful coverage of possible observations. They cater for the infinite data types of *Circus* models, with operations specified in the Z predicative style. The symbolic tests, along with the symbolic traces and acceptance sets used to define them, are a prerequisite for proposing and justifying test-data generation techniques in any language combining control and complex data types. They specify the constraint-solving problems that need to be addressed.

On the other hand, complementary selection criteria that have been widely explored are based on coverage of the syntactic structure of the specification: actions, transitions, paths that link variable assignments and uses, and so on. The labelled transition system defined by the *Circus* operational semantics, however, abstracts from this structure, including from the particular way in which variables are used. Moreover, it includes transitions that do not correspond to observable behaviour; their coverage is unlikely to be interesting for testing.

In this paper, we present a new labelled transition system for *Circus* that is appropriate for the definition of specification-based coverage criteria, and the associated algorithms for test-case generation. We define the new system in terms of two other new transition relations, which we also present here. We briefly discuss the soundness of the new transition rules, but leave a complete account as future work. For illustration purposes, we explain how we can use the new transition system to define a definition-use selection criterion [13].

Section 2 provides an introduction to *Circus*, its operational semantics, and testing theory. The specification-oriented transition system is described in Section 3. Its use in testing is the subject of Section 4. Soundness is discussed in Section 5. Finally, we present our conclusions in Section 6. Appendix A reproduces the rules of the operational semantics used in discussions and examples.

2 *Circus*, its operational semantics, and testing

The UTP has been the basis for a now ten-year-old research agenda on the development of the *Circus* family of languages based on a combination of Z and CSP. There have been extensions to cater for time [32], synchronicity [4], mobility [33], pointers [16], and object-orientation [8, 29]. We give here a brief description of the original *Circus* language, and its operational semantics [36]. Our results, however, are a starting point to consider coverage of specifications for all *Circus* extensions. They are all justified using UTP theories.

2.1 *Circus* notation

A *Circus* model is formed by a sequence of paragraphs, like in Z [37], but they can define channels and processes, like in CSP and its machine-readable version (CSP-M) [28]. Figure 1 presents a model of a resource manager. Its first paragraph introduces a given set *Resource* including the valid resources. The second paragraph declares two channels: *insert* is used to request the addition of a resource in the pool, and *get* to request a resource from the pool. The last paragraph is a basic (or explicit) definition for a process called *ResourceManager*.

A basic process definition is itself formed by a sequence of paragraphs. The first paragraph of *ResourceManager* is a Z schema *RM* marked as the **state** definition. *Circus* processes have a private state defined using Z, and interact with each other and their environment using channels, like CSP processes.

The state of *ResourceManager* includes two components: a *pool* of resources, and a *cache* that records a resource ready for delivery. The state invariant requires that the cached resource is not in the pool as well.

Operations over the state can be defined by schemas just like in Z. For instance, the schema *Cache* specifies an operation that caches a resource, if the pool is not empty. The schema *Cache* includes the schema ΔRM to bring into scope the names of the state components defined in *RM* and their dashed counterparts to represent the state after the execution of *Cache*.

State operations are called actions in *Circus*, and can also be defined using Morgan's specification statements [22] or guarded commands from Dijkstra's language [12]. The operation *Insert* in our example, for instance, is defined by an assignment. It adds a resource $r?$ given as input to the pool.

CSP constructs can also be used to specify actions. For instance, the resource manager has two components: a *CacheManager* and a *PoolManager*, specified by separate actions. *CacheManager* accepts requests for a resource through the channel *get*. When such a request occurs, the cache becomes empty and the manager terminates. The *PoolManager*, on the other hand, accepts requests to insert a resource in the pool, which is carried by *Insert*. It also monitors requests for a resource (through *get*). When this happens, if the pool is not empty, the manager terminates, otherwise, it waits for an element to be inserted in the pool before terminating. The specification of *PoolManager* combines an assignment, the action **Skip**, which terminates immediately, without changing the state, a

$[Resource]$

channel $insert, get : Resource$

process $ResourceManager \mathrel{\widehat=} \textbf{begin}$

state RM

$pool : \mathbb{P}\, Resource$
$cache : Resource$

$cache \notin pool$

Init

RM'

$pool' = \varnothing$

Cache

ΔRM

$pool \neq \varnothing$
$pool' = pool \setminus \{cache'\}$

$Insert \mathrel{\widehat=} r? : Resource \bullet pool := pool \cup \{r?\}$

$CacheManager \mathrel{\widehat=} get!cache \rightarrow \textbf{Skip}$

$$PoolManager \mathrel{\widehat=} \left(\mu X \bullet \left(\begin{array}{l} insert?r \rightarrow Insert(r);\ X \\ \Box \\ get?x \rightarrow (\textbf{if}\ pool \neq \varnothing \rightarrow \textbf{Skip}\ [\!]\ pool = \varnothing \rightarrow insert?r \rightarrow Insert(r)) \end{array}\right)\right)$$

$\bullet\ Init;$
$(\mu X \bullet (CacheManager \llbracket \{cache\} \mid \{\!|\, get \,|\!\} \mid \{pool\} \rrbracket PoolManager);\ Cache;\ X)$
end

Fig. 1. Resource manager in *Circus*

schema operation, a conditional (in Dijkstra's style), and an external choice ($\Box$). Z and CSP constructs are intermixed freely in an action definition.

A main action at the end defines the behaviour of the *ResourceManager*. After a call to the initialisation operation *Init*, a parallel composition combines the *CacheManager* and the *PoolManager*. When the parallelism terminates, the *Cache* operation updates the cache to make a resource available.

Like in CSP, the parallel operator defines a synchronisation channel set: communications through channels in this set require agreement of both parallel actions. In our example, *get* is in the synchronisation set. To avoid race conditions, the parallel operator also associates with each action a partition of the variables in scope over which it has write control. Both parallel actions can access the value of the state before the parallelism starts. Both can write to all state components. An update, however, only becomes visible to other actions after the parallelism terminates, and then only if the action has write control over the changed variable. In our example, the *CacheManager* has control over *cache*, and *PoolManager* over *pool*. (In fact, *CacheManager* does not change *cache*, but we require the name sets to be a partition of the state.)

Processes can also be defined by composition using CSP constructs. For example, in a distributed setting, we can have two resource managers available.

$$\textbf{process}\ Resources \mathrel{\widehat=} ResourceManager \mid\mid\mid ResourceManager$$

In this case, we have two copies of a *ResourceManager*, each with its own private state. A request to *insert* or *get* a resource is responded by either of them; the choice is nondeterministic. Nondeterminism in *Circus* may arise from data operations, like in Z, or from parallelism, like in CSP. We also have the explicit operator for nondeterministic choice of CSP. In our example, the data operation *Cache* is nondeterministic, as is the process *Resources* above.

A full account of *Circus* and its denotational semantics, including the UTP theory that underpins it, is given in [25]. The *Circus* operational semantics [36] is briefly discussed and illustrated in the next section.

2.2 *Circus* operational semantics

As usual, the operational semantics of *Circus* is based on a transition relation that associates configurations. For processes, the configurations are processes themselves. For actions, the configurations are triples. The first component is a constraint over symbolic variables used to define labels and the state. The second component is a total assignment in the UTP theory of relations of symbolic variables to variables. The last component is an action.

The constraints in the configurations are texts that denote predicates (over symbolic variables). Like in the UTP, we use typewriter font for pieces of text. The syntax used to define them is that of the UTP relational theory, and of *Circus* predicates, which are basically Z predicates [23].

State assignments are expressed using the UTP notation $x := e$ for relational assignments. They can also include declarations and undeclaration of variables

using the UTP constructs $\mathbf{var}\, x$ and $\mathbf{end}\, x$. The declaration of a variable is immediately followed by an assignment of a symbolic variable to it, so that state assignments are deterministic programs that define a specific value (represented by a symbolic variable) for all variables in scope. We use the notation $\mathbf{var}\, x := e$ as an abbreviation for $\mathbf{var}\, x;\ x := e$. It is the combination of the constraint over symbolic variables, and the state assignment of symbolic variables to all variables in scope that, together, specify the state of a configuration.

To give the operational semantics of a process, we use a novel construct for a basic process. It records the current local state of a process using a constraint and a state assignment. The first transition rule for processes shown below introduces the record of the local state. It is characterised using a (list of) fresh symbolic variable(s) $\mathtt{w_0}$. The constraint defines that $\mathtt{w_0}$ is (are) of the appropriate type(s), and in the state assignment $\mathtt{w_0}$ is assigned to the state component(s) $\mathtt{x}$. In all transition rules, the symbolic variables introduced are assumed to be fresh.

$$\left(\begin{array}{l}\mathtt{begin}\\ \quad \mathtt{state}\ [\mathtt{x} : \mathtt{T}]\\ \quad \bullet\ \mathtt{A}\\ \mathtt{end}\end{array}\right) \xrightarrow{\epsilon} \left(\begin{array}{l}\mathtt{begin}\\ \quad \mathtt{state}\ [\mathtt{x} : \mathtt{T}] \mid \mathtt{loc}\ (\mathtt{w_0} \in \mathtt{T} \mid \mathtt{x} := \mathtt{w_0})\\ \quad \bullet\ \mathtt{A}\\ \mathtt{end}\end{array}\right)$$

The semantics of composed processes is defined by providing a corresponding basic process [23]. We, therefore, do not consider them here. The complete set of transition rules is in [36]; some are presented below and in Appendix A.

The second transition rule for processes, which we omit here for conciseness, applies to the extended form of a basic process. The rule allows a process to evolve in accordance with the evolution of its main action in the state defined by the `loc` clause. We therefore focus on the transition relation for actions.

The rule for designs (which are a simplified form of specification statement) is below. The hypothesis requires the constraint to hold, the precondition to hold in the current state s, and the design to be feasible. In this case, evolution to `Skip` is silent (not labelled, or labelled by ϵ). The constraint is strengthened by introducing fresh symbolic variables $\mathtt{w_0}$ that satisfy the postcondition, and the state is updated by assigning $\mathtt{w_0}$ to all variables in scope. The state `s` is not completely discarded, since it may contain variable declarations.

$$\frac{c \wedge (s;\ p) \wedge (\exists\, v' \bullet s;\ Q)}{(\mathtt{c} \mid \mathtt{s} \models \mathtt{p} \vdash \mathtt{Q}) \xrightarrow{\epsilon} (\mathtt{c} \wedge (\mathtt{s};\ \mathtt{Q}\,[\mathtt{w_0}/\mathtt{v'}]) \mid \mathtt{s};\ \mathtt{v} := \mathtt{w_0} \models \mathtt{Skip})} \quad v' = out\alpha s$$

Another rule states that, if the precondition does not hold, the design evolves to **Chaos**, the action that diverges immediately.

The evolution of an output prefixing $\mathtt{d!e} \rightarrow \mathtt{A}$, an action that outputs the value of the expression `e` through channel `d` and then behaves like `A`, is labelled. The label $\mathtt{d.w_0}$ involves the fresh constant $\mathtt{w_0}$; the new constraint defines its value to

be that of $\mathtt{e}$ in the current state $\mathtt{s}$. The remaining action is $\mathtt{A}$.

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

The transition rule for an input prefixing $\mathtt{d?x} \to \mathtt{A}$ is as follows.

$$\frac{c \wedge T \neq \varnothing \qquad x \notin \alpha s}{(\mathtt{c} \mid \mathtt{s} \models \mathtt{d?x} : \mathtt{T} \to \mathtt{A}) \xrightarrow{\mathtt{d?w_0}} (\mathtt{c} \wedge \mathtt{w_0} \in \mathtt{T} \mid \mathtt{s};\ \mathtt{var\,x} := \mathtt{w_0} \models \mathtt{let\ x} \bullet \mathtt{A})}$$

The label is $\mathtt{d?w_0}$. In the new the state, $\mathtt{x}$ is declared and assigned $\mathtt{w_0}$. The only restriction on $\mathtt{w_0}$ is that it has the same type as $\mathtt{d}$. The remaining action $\mathtt{let\ x} \bullet \mathtt{A}$ records the fact that $\mathtt{x}$ is in scope in $\mathtt{A}$ as a local variable. The construct $\mathtt{let\ x} \bullet \mathtt{A}$ has been introduced specifically for use in the operational semantics. When $\mathtt{A}$ terminates, a rule for $\mathtt{let\ x} \bullet \mathtt{Skip}$ closes the scope of $\mathtt{x}$ in the state and removes the $\mathtt{let\ x}$ declaration. This is Rule (8) in Appendix A.

The transition rules for sequences $\mathtt{A_1};\ \mathtt{B}$ are standard. Evolution of $\mathtt{A_1}$ leads to evolution of the sequence. When it terminates, a rule for $\mathtt{Skip};\ \mathtt{B}$ allows a silent transition to $\mathtt{B}$, thus removing the sequence.

$$\frac{(\mathtt{c_1} \mid \mathtt{s_1} \models \mathtt{A_1}) \xrightarrow{\mathtt{l}} (\mathtt{c_2} \mid \mathtt{s_2} \models \mathtt{A_2})}{(\mathtt{c_1} \mid \mathtt{s_1} \models \mathtt{A_1};\ \mathtt{B}) \xrightarrow{\mathtt{l}} (\mathtt{c_2} \mid \mathtt{s_2} \models \mathtt{A_2};\ \mathtt{B})} \qquad \frac{c}{(\mathtt{c} \mid \mathtt{s} \models \mathtt{Skip};\ \mathtt{A}) \xrightarrow{\epsilon} (\mathtt{c} \mid \mathtt{s} \models \mathtt{A})}$$

For an internal choice $\mathtt{A_1} \sqcap \mathtt{A_2}$, silent transitions are available to either $\mathtt{A_1}$ or $\mathtt{A_2}$ (in a configuration with the same constraint and state assignment).

$$\frac{c}{(\mathtt{c} \mid \mathtt{s} \models \mathtt{A_1} \sqcap \mathtt{A_2}) \xrightarrow{\epsilon} (\mathtt{c} \mid \mathtt{s} \models \mathtt{A_1})} \qquad \frac{c}{(\mathtt{c} \mid \mathtt{s} \models \mathtt{A_1} \sqcap \mathtt{A_2}) \xrightarrow{\epsilon} (\mathtt{c} \mid \mathtt{s} \models \mathtt{A_2})}$$

The treatment of parallelism is more subtle. We introduce an extra form of action $\mathsf{par}\ s \mid x \bullet A$ that records the local state s of the parallel action A, which has write control over the variables in x. The first transition rule for a parallelism defines a silent transition that rewrites it in terms of this new construct.

The rule below allows evolutions of the first parallel action $\mathtt{A_1}$ that are either silent or do not involve a channel in the synchronisation set to be reflected in

the parallelism. A similar omitted rule considers independent evolutions of $\mathtt{A}_2$.

$$\frac{(\mathtt{c} \mid \mathtt{s}_1 \models \mathtt{A}_1) \xrightarrow{\mathtt{l}} (\mathtt{c}_3 \mid \mathtt{s}_3 \models \mathtt{A}_3) \quad \mathtt{l} = \epsilon \vee \mathrm{chan}\,\mathtt{l} \notin cs}{\left(\begin{array}{l}\mathtt{c} \mid \mathtt{s} \\ \models \\ \left(\begin{array}{c}(\mathtt{par}\ \mathtt{s}_1 \mid \mathtt{x}_1 \bullet \mathtt{A}_1) \\ [\![\mathtt{cs}]\!] \\ (\mathtt{par}\ \mathtt{s}_2 \mid \mathtt{x}_2 \bullet \mathtt{A}_2)\end{array}\right)\end{array}\right) \xrightarrow{\mathtt{l}} \left(\begin{array}{l}\mathtt{c}_3 \mid \mathtt{s} \\ \models \\ \left(\begin{array}{c}(\mathtt{par}\ \mathtt{s}_3 \mid \mathtt{x}_1 \bullet \mathtt{A}_3) \\ [\![\mathtt{cs}]\!] \\ (\mathtt{par}\ \mathtt{s}_2 \mid \mathtt{x}_2 \bullet \mathtt{A}_2)\end{array}\right)\end{array}\right)}$$

The next rule is for when the parallel actions can evolve by synchronising. In particular, $\mathtt{A}_1$ can carry out an input $\mathtt{d}?\mathtt{w}_1$, and $\mathtt{A}_2$ an output $\mathtt{d}!\mathtt{w}_2$, where $\mathtt{d}$ is a channel in the synchronisation set, and the values communicated are equal. The transition rule establishes that, in this case, the parallelism as a whole actually performs an output. The new constraint records the restriction that $\mathtt{w}_1 = \mathtt{w}_2$.

$$\frac{(\mathtt{c} \mid \mathtt{s}_1 \models \mathtt{A}_1) \xrightarrow{\mathtt{d}?\mathtt{w}_1} (\mathtt{c}_3 \mid \mathtt{s}_3 \models \mathtt{A}_3) \qquad (\mathtt{c} \mid \mathtt{s}_2 \models \mathtt{A}_2) \xrightarrow{\mathtt{d}!\mathtt{w}_2} (\mathtt{c}_4 \mid \mathtt{s}_4 \models \mathtt{A}_4) \qquad \mathtt{d} \in cs \quad c_3 \wedge c_4 \wedge w_1 = w_2}{\left(\begin{array}{l}\mathtt{c} \mid \mathtt{s} \\ \models \\ \left(\begin{array}{c}(\mathtt{par}\ \mathtt{s}_1 \mid \mathtt{x}_1 \bullet \mathtt{A}_1) \\ [\![\mathtt{cs}]\!] \\ (\mathtt{par}\ \mathtt{s}_2 \mid \mathtt{x}_2 \bullet \mathtt{A}_2)\end{array}\right)\end{array}\right) \xrightarrow{\mathtt{d}!\mathtt{w}_2} \left(\begin{array}{l}\mathtt{c}_3 \wedge \mathtt{c}_4 \wedge \mathtt{w}_1 = \mathtt{w}_2 \mid \mathtt{s} \\ \models \\ \left(\begin{array}{c}(\mathtt{par}\ \mathtt{s}_3 \mid \mathtt{x}_1 \bullet \mathtt{A}_3) \\ [\![\mathtt{cs}]\!] \\ (\mathtt{par}\ \mathtt{s}_4 \mid \mathtt{x}_2 \bullet \mathtt{A}_4)\end{array}\right)\end{array}\right)}$$

Similar rules apply when $\mathtt{A}_1$ can output and $\mathtt{A}_2$ input, or when both $\mathtt{A}_1$ and $\mathtt{A}_2$ can output. When they can both input, the parallelism also performs an input. We refer to the Appendix A for an account of all transition rules for parallelism.

Perhaps the most interesting rule is the one that applies when both parallel actions have terminated. In this case, the parallelism terminates.

$$\frac{c}{\left(\begin{array}{l}\mathtt{c} \mid \mathtt{s} \\ \models \\ \left(\begin{array}{c}(\mathtt{par}\ \mathtt{s}_1 \mid \mathtt{x}_1 \bullet \mathtt{Skip}) \\ [\![\mathtt{cs}]\!] \\ (\mathtt{par}\ \mathtt{s}_2 \mid \mathtt{x}_2 \bullet \mathtt{Skip})\end{array}\right)\end{array}\right) \xrightarrow{\epsilon} (\mathtt{c} \mid (\exists\, \mathtt{x}_2' \bullet \mathtt{s}_1) \wedge (\exists\, \mathtt{x}_1' \bullet \mathtt{s}_2) \models \mathtt{Skip})}$$

The state after the parallelism is defined by composing the local states of the parallel action. We keep from the local state $\mathtt{s}_1$ of the first action only the changes to the variables in its name set $\mathtt{x}_1$. This is achieved by hiding (quantifying) the final value of the variables in the complement set $\mathtt{x}_2$. The same applies for the changes in $\mathtt{s}_2$. The conjunction of the quantifications defines the new state. We observe that, alternatively, we can define the new state as $\mathtt{s}_1;\ \mathtt{end}\,\mathtt{x}_2 \wedge \mathtt{s}_2;\ \mathtt{end}\,\mathtt{x}_1$.

Rules for external choice require similar considerations. Actions in an external choice can evolve independently, with local access to all variables, until an event decides the choice, and consequently, makes the local changes global.

For a hiding $\mathtt{A}_1 \setminus \mathtt{cs}$, the rules allow evolution of $\mathtt{A}_1$ to lead to evolution of the hiding itself. In the rule below, evolution does not involve a hidden channel, so the label for the hiding transition is that for the $\mathtt{A}_1$ transition.

$$\frac{(\mathtt{c}_1 \mid \mathtt{s}_1 \models \mathtt{A}_1) \xrightarrow{\mathtt{l}} (\mathtt{c}_2 \mid \mathtt{s}_2 \models \mathtt{A}_2) \quad \text{chan } \mathtt{l} \notin cs}{(\mathtt{c}_1 \mid \mathtt{s}_1 \models \mathtt{A}_1 \setminus \mathtt{cs}) \xrightarrow{\mathtt{l}} (\mathtt{c}_2 \mid \mathtt{s}_2 \models \mathtt{A}_2 \setminus \mathtt{cs})}$$

If, on the other hand, $\mathtt{A}_1$ can communicate on a hidden channel, the corresponding evolution of the hiding is silent. This is defined by Rule (20) in Appendix A. Finally, Rule (21) specifies that if $\mathtt{A}_1$ terminates, so does the hiding.

Example 1. We consider the action defined below, in the context of a process that has a state with components x and y, of type $\mathbb{Z}$, for instance. Channels *inp* and *out* also of type $\mathbb{Z}$ are in scope and used in E.

$$E \mathrel{\widehat=} x := 2;\ (y > x \ \&\ out!(y - x) \rightarrow \mathbf{Skip} \sqcap inp?z \rightarrow \mathbf{Stop});\ x := y$$

As suggested by the transition rule for processes, we consider the transitions from a state characterised by $(\mathtt{w}_0 \in \mathbb{Z} \land \mathtt{w}_1 \in \mathbb{Z} \mid \mathtt{x}, \mathtt{y} := \mathtt{w}_0, \mathtt{w}_1)$. We can justify the following transitions using the *Circus* operational semantics described above. The rule numbers mentioned refer to the list in Appendix A.

$$(\mathtt{w}_0 \in \mathbb{Z} \land \mathtt{w}_1 \in \mathbb{Z} \mid \mathtt{x}, \mathtt{y} := \mathtt{w}_0, \mathtt{w}_1 \models \mathtt{E})$$

$\longrightarrow$ [Rules (2) and (9)]

$$\left(\begin{array}{l} \mathtt{w}_0 \in \mathbb{Z} \land \mathtt{w}_1 \in \mathbb{Z} \land \mathtt{w}_2 = 2 \mid \mathtt{x}, \mathtt{y} := \mathtt{w}_2, \mathtt{w}_1 \\ \models \\ \mathtt{Skip};\ (\mathtt{y} > \mathtt{x}\ \&\ \mathtt{out}!(\mathtt{y} - \mathtt{x}) \rightarrow \mathtt{Skip} \sqcap \mathtt{inp}?\mathtt{z} \rightarrow \mathtt{Stop});\ \mathtt{x} := \mathtt{y} \end{array}\right)$$

$\longrightarrow$ [Rule (10)]

$$\left(\begin{array}{l} \mathtt{w}_0 \in \mathbb{Z} \land \mathtt{w}_1 \in \mathbb{Z} \land \mathtt{w}_2 = 2 \mid \mathtt{x}, \mathtt{y} := \mathtt{w}_2, \mathtt{w}_1 \\ \models \\ (\mathtt{y} > \mathtt{x}\ \&\ \mathtt{out}!(\mathtt{y} - \mathtt{x}) \rightarrow \mathtt{Skip} \sqcap \mathtt{inp}?\mathtt{z} \rightarrow \mathtt{Stop});\ \mathtt{x} := \mathtt{y} \end{array}\right)$$

At this point two rules for internal choice apply, corresponding to the two choices available. We pursue the first below, and the second afterwards.

$\longrightarrow$ [Rules (11) and (9)]

$$\left(\begin{array}{l} \mathtt{w}_0 \in \mathbb{Z} \land \mathtt{w}_1 \in \mathbb{Z} \land \mathtt{w}_2 = 2 \mid \mathtt{x}, \mathtt{y} := \mathtt{w}_2, \mathtt{w}_1 \\ \models \\ (\mathtt{y} > \mathtt{x}\ \&\ \mathtt{out}!(\mathtt{y} - \mathtt{x}) \rightarrow \mathtt{Skip});\ \mathtt{x} := \mathtt{y} \end{array}\right)$$

$\longrightarrow$ [Rules (12) and (9)]

$$\left(\begin{array}{l} \mathtt{w}_0 \in \mathbb{Z} \land \mathtt{w}_1 \in \mathbb{Z} \land \mathtt{w}_2 = 2 \land \mathtt{w}_1 > \mathtt{w}_2 \mid \mathtt{x}, \mathtt{y} := \mathtt{w}_2, \mathtt{w}_1 \\ \models \\ (\mathtt{out!}(\mathtt{y} - \mathtt{x}) \rightarrow \mathtt{Skip});\ \mathtt{x} := \mathtt{y} \end{array}\right)$$

$\xrightarrow{\mathtt{out!w_3}}$ [Rules (4) and (9)]

$$\left(\begin{array}{l} \mathtt{w}_0 \in \mathbb{Z} \land \mathtt{w}_1 \in \mathbb{Z} \land \mathtt{w}_2 = 2 \land \mathtt{w}_1 > \mathtt{w}_2 \land \mathtt{w}_3 = \mathtt{w}_1 - \mathtt{w}_2 \mid \mathtt{x}, \mathtt{y} := \mathtt{w}_2, \mathtt{w}_1 \\ \models \\ \mathtt{Skip};\ \mathtt{x} := \mathtt{y} \end{array}\right)$$

$\longrightarrow$ [Rule (10)]

$$\left(\begin{array}{l} \mathtt{w}_0 \in \mathbb{Z} \land \mathtt{w}_1 \in \mathbb{Z} \land \mathtt{w}_2 = 2 \land \mathtt{w}_1 > \mathtt{w}_2 \land \mathtt{w}_3 = \mathtt{w}_1 - \mathtt{w}_2 \mid \mathtt{x}, \mathtt{y} := \mathtt{w}_2, \mathtt{w}_1 \\ \models \\ \mathtt{x} := \mathtt{y} \end{array}\right)$$

$\longrightarrow$ [Rule (2)]

$$\left(\begin{array}{l} \mathtt{w}_0 \in \mathbb{Z} \land \mathtt{w}_1 \in \mathbb{Z} \land \mathtt{w}_2 = 2 \land \mathtt{w}_1 > \mathtt{w}_2 \land \mathtt{w}_3 = \mathtt{w}_1 - \mathtt{w}_2 \land \mathtt{w}_4 = \mathtt{w}_2 \mid \mathtt{x}, \mathtt{y} := \mathtt{w}_4, \mathtt{w}_1 \\ \models \\ \mathtt{Skip} \end{array}\right)$$

Considering the second option of the internal choice, we can proceed as follows.

$\longrightarrow$ [Rules (11) and (9)]

$$\left(\begin{array}{l} \mathtt{w}_0 \in \mathbb{Z} \land \mathtt{w}_1 \in \mathbb{Z} \land \mathtt{w}_2 = 2 \mid \mathtt{x}, \mathtt{y} := \mathtt{w}_2, \mathtt{w}_1 \\ \models \\ (\mathtt{inp?z} \rightarrow \mathtt{Stop});\ \mathtt{x} := \mathtt{y} \end{array}\right)$$

$\xrightarrow{\mathtt{inp?w_3}}$ [Rules (5) and (9)]

$$\left(\begin{array}{l} \mathtt{w}_0 \in \mathbb{Z} \land \mathtt{w}_1 \in \mathbb{Z} \land \mathtt{w}_2 = 2 \land \mathtt{w}_3 \in \mathbb{Z} \mid \mathtt{x}, \mathtt{y} := \mathtt{w}_2, \mathtt{w}_1;\ \mathtt{var\,z} := \mathtt{w}_3 \\ \models \\ (\mathtt{let\ z} \bullet \mathtt{Stop});\ \mathtt{x} := \mathtt{y} \end{array}\right)$$

From here, we cannot proceed, as there are no transition rules for `Stop`.

All transitions above are valid when the associated constraints are satisfied. □

Example 2. We consider the action defined below, in the context of a process that has a state with a component x of type $\mathbb{Z}$. Channels *inpA*, *inpB*, *int*, and *out*, also of type $\mathbb{Z}$, are in scope and used in *PA*.

$$PA \mathrel{\widehat{=}} \left(\begin{array}{l} x := 2;\ \left(\begin{array}{l} (inpA?y \rightarrow int!y \rightarrow out!(y - x) \rightarrow \mathbf{Skip}) \\ \quad [\![\{\!| int |\!\}]\!] \\ (inpB?z_1 \rightarrow int?z_2 \rightarrow z_1 > z_2 \mathbin{\&} out!(z_1 - x) \rightarrow \mathbf{Skip}) \end{array}\right) \end{array}\right)$$

Strictly speaking, we would need to define the sets of names of variables that can be updated by each of the parallel actions. In this simple example, however, they update no variables, so we omit these sets.

We consider below the transitions from the state $(\mathtt{w}_0 \in \mathbb{Z} \mid \mathtt{x} := \mathtt{w}_0)$. We can justify the following transitions using the *Circus* operational semantics.

$$(\mathtt{w}_0 \in \mathbb{Z} \mid \mathtt{x} := \mathtt{w}_0 \models \mathtt{PA})$$

$\longrightarrow$ [Rules (2) and (9)]

$$\left(\begin{array}{l} \mathtt{w}_0 \in \mathbb{Z} \land \mathtt{w}_1 = 2 \mid \mathtt{x} := \mathtt{w}_1 \\ \models \\ \quad \mathtt{Skip}; \left(\begin{array}{l} (\mathtt{inpA?y} \rightarrow \mathtt{int!y} \rightarrow \mathtt{out!}(\mathtt{y} - \mathtt{x}) \rightarrow \mathtt{Skip}) \\ \quad [\![\{\!| \mathtt{int} |\!\}]\!] \\ (\mathtt{inpB?z}_1 \rightarrow \mathtt{int?z}_2 \rightarrow \mathtt{z}_1 > \mathtt{z}_2 \,\&\, \mathtt{out!}(\mathtt{z}_1 - \mathtt{x}) \rightarrow \mathtt{Skip}) \end{array}\right) \end{array}\right)$$

$\longrightarrow$ [Rule (10)]

$$\left(\begin{array}{l} \mathtt{w}_0 \in \mathbb{Z} \land \mathtt{w}_1 = 2 \mid \mathtt{x} := \mathtt{w}_1 \\ \models \\ \left(\begin{array}{l} (\mathtt{inpA?y} \rightarrow \mathtt{int!y} \rightarrow \mathtt{out!}(\mathtt{y} - \mathtt{x}) \rightarrow \mathtt{Skip}) \\ \quad [\![\{\!| \mathtt{int} |\!\}]\!] \\ (\mathtt{inpB?z}_1 \rightarrow \mathtt{int?z}_2 \rightarrow \mathtt{z}_1 > \mathtt{z}_2 \,\&\, \mathtt{out!}(\mathtt{z}_1 - \mathtt{x}) \rightarrow \mathtt{Skip}) \end{array}\right) \end{array}\right)$$

$\longrightarrow$ [Rule (13)]

$$\left(\begin{array}{l} \mathtt{w}_0 \in \mathbb{Z} \land \mathtt{w}_1 = 2 \mid \mathtt{x} := \mathtt{w}_1 \\ \models \\ \left(\begin{array}{l} \mathtt{par\ x} := \mathtt{w}_1 \bullet \mathtt{inpA?y} \rightarrow \mathtt{int!y} \rightarrow \mathtt{out!}(\mathtt{y} - \mathtt{x}) \rightarrow \mathtt{Skip} \\ \quad [\![\{\!| \mathtt{int} |\!\}]\!] \\ \mathtt{par\ x} := \mathtt{w}_1 \bullet \mathtt{inpB?z}_1 \rightarrow \mathtt{int?z}_2 \rightarrow \mathtt{z}_1 > \mathtt{z}_2 \,\&\, \mathtt{out!}(\mathtt{z}_1 - \mathtt{x}) \rightarrow \mathtt{Skip} \end{array}\right) \end{array}\right)$$

Now, there are two rules that are applicable (Rules (15) and (16)), reflecting the fact that either of the parallel actions can evolve independently. So, we can have the following sequence of transitions if the left-hand action evolves first.

$\xrightarrow{\mathtt{inpA?w}_2}$ [Rules (5) and (15)]

$$\left(\begin{array}{l} \mathtt{w}_0 \in \mathbb{Z} \land \mathtt{w}_1 = 2 \land \mathtt{w}_2 \in \mathbb{Z} \mid \mathtt{x} := \mathtt{w}_1 \\ \models \\ \left(\begin{array}{l} \mathtt{par\ x} := \mathtt{w}_1; \ \mathtt{var\,y} := \mathtt{w}_2 \bullet (\mathtt{let\ y} \bullet \mathtt{int!y} \rightarrow \mathtt{out!}(\mathtt{y} - \mathtt{x}) \rightarrow \mathtt{Skip}) \\ \quad [\![\{\!| \mathtt{int} |\!\}]\!] \\ \mathtt{par\ x} := \mathtt{w}_1 \bullet \mathtt{inpB?z}_1 \rightarrow \mathtt{int?z}_2 \rightarrow \mathtt{z}_1 > \mathtt{z}_2 \,\&\, \mathtt{out!}(\mathtt{z}_1 - \mathtt{x}) \rightarrow \mathtt{Skip} \end{array}\right) \end{array}\right)$$

$\xrightarrow{\mathtt{inpB?w}_3}$ [Rules (5) and (16)]

$$\left(\begin{array}{l} \mathtt{w}_0 \in \mathbb{Z} \land \mathtt{w}_1 = 2 \land \mathtt{w}_2 \in \mathbb{Z} \land \mathtt{w}_3 \in \mathbb{Z} \mid \mathtt{x} := \mathtt{w}_1 \\ \models \\ \left(\begin{array}{l} \mathtt{par\ x} := \mathtt{w}_1; \ \mathtt{var\,y} := \mathtt{w}_2 \bullet (\mathtt{let\ y} \bullet \mathtt{int!y} \rightarrow \mathtt{out!}(\mathtt{y} - \mathtt{x}) \rightarrow \mathtt{Skip}) \\ \quad [\![\{\!| \mathtt{int} |\!\}]\!] \\ \left(\begin{array}{l} \mathtt{par\ x} := \mathtt{w}_1; \ \mathtt{var\,z}_1 := \mathtt{w}_3 \bullet \\ \quad \mathtt{let\ z}_1 \bullet \mathtt{int?z}_2 \rightarrow \mathtt{z}_1 > \mathtt{z}_2 \,\&\, \mathtt{out!}(\mathtt{z}_1 - \mathtt{x}) \rightarrow \mathtt{Skip} \end{array}\right) \end{array}\right) \end{array}\right)$$

If the right-hand action evolves first, we have the following transitions. We choose the names of the symbolic variables carefully, so that the same communicated

values are represented by variables of the same name. In our use of the operational semantics to define traces, initials, acceptances, and tests [6], this careful choice is guided and fixed by an alphabet of symbolic variables.

$$\xrightarrow{\texttt{inpB?}\mathtt{w}_3} \qquad \text{[Rules (5) and (16)]}$$

$$\left(\begin{array}{l} \mathtt{w}_0 \in \mathbb{Z} \land \mathtt{w}_1 = 2 \land \mathtt{w}_3 \in \mathbb{Z} \mid \mathtt{x} := \mathtt{w}_1 \\ \models \\ \left(\begin{array}{l} \texttt{par}\ \mathtt{x} := \mathtt{w}_1 \bullet \texttt{inpA?}\mathtt{y} \to \texttt{int!}\mathtt{y} \to \texttt{out!}(\mathtt{y} - \mathtt{x}) \to \texttt{Skip} \\ \quad [\![\{\!| \texttt{int} |\!\}]\!] \\ \left(\begin{array}{l} \texttt{par}\ \mathtt{x} := \mathtt{w}_1;\ \texttt{var}\,\mathtt{z}_1 := \mathtt{w}_3 \bullet \\ \quad \texttt{let}\ \mathtt{z}_1 \bullet \texttt{int?}\mathtt{z}_2 \to \mathtt{z}_1 > \mathtt{z}_2\ \&\ \texttt{out!}(\mathtt{z}_1 - \mathtt{x}) \to \texttt{Skip} \end{array}\right) \end{array}\right) \end{array}\right)$$

$$\xrightarrow{\texttt{inpA?}\mathtt{w}_2} \qquad \text{[Rules (5) and (15)]}$$

$$\left(\begin{array}{l} \mathtt{w}_0 \in \mathbb{Z} \land \mathtt{w}_1 = 2 \land \mathtt{w}_2 \in \mathbb{Z} \land \mathtt{w}_3 \in \mathbb{Z} \mid \mathtt{x} := \mathtt{w}_1 \\ \models \\ \left(\begin{array}{l} \texttt{par}\ \mathtt{x} := \mathtt{w}_1;\ \texttt{var}\,\mathtt{y} := \mathtt{w}_2 \bullet (\texttt{let}\ \mathtt{y} \bullet \texttt{int!}\mathtt{y} \to \texttt{out!}(\mathtt{y} - \mathtt{x}) \to \texttt{Skip}) \\ \quad [\![\{\!| \texttt{int} |\!\}]\!] \\ \left(\begin{array}{l} \texttt{par}\ \mathtt{x} := \mathtt{w}_1;\ \texttt{var}\,\mathtt{z}_1 := \mathtt{w}_3 \bullet \\ \quad \texttt{let}\ \mathtt{z}_1 \bullet \texttt{int?}\mathtt{z}_2 \to \mathtt{z}_1 > \mathtt{z}_2\ \&\ \texttt{out!}(\mathtt{z}_1 - \mathtt{x}) \to \texttt{Skip} \end{array}\right) \end{array}\right) \end{array}\right)$$

The configurations reached in both options are the same. (If we did not choose the names of the symbolic variables appropriately, there would be syntactic differences in the text of the constraint and state assignment, arising (just) from the differentiated use of fresh names. For the sake of simplicity, we are choosing the names in an adequate way as explained before. With the support of simple pattern matching facilities, a tool can identify the commonality in any case.)

The next transition rule that applies is that for synchronisation of parallel actions, when we have a matching input and output.

$$\xrightarrow{\texttt{int!}\mathtt{w}_5} \qquad \text{[Rules (4), (7), (5), and (17)]}$$

$$\left(\begin{array}{l} \mathtt{w}_0 \in \mathbb{Z} \land \mathtt{w}_1 = 2 \land \mathtt{w}_2 \in \mathbb{Z} \land \mathtt{w}_3 \in \mathbb{Z} \land \mathtt{w}_4 = \mathtt{w}_2 \land \mathtt{w}_5 \in \mathbb{Z} \land \mathtt{w}_4 = \mathtt{w}_5 \mid \mathtt{x} := \mathtt{w}_1 \\ \models \\ \left(\begin{array}{l} \texttt{par}\ \mathtt{x} := \mathtt{w}_1;\ \texttt{var}\,\mathtt{y} := \mathtt{w}_2 \bullet (\texttt{let}\ \mathtt{y} \bullet \texttt{out!}(\mathtt{y} - \mathtt{x}) \to \texttt{Skip}) \\ \quad [\![\{\!| \texttt{int} |\!\}]\!] \\ \left(\begin{array}{l} \texttt{par}\ \mathtt{x} := \mathtt{w}_1;\ \texttt{var}\,\mathtt{z}_1, \mathtt{z}_2 := \mathtt{w}_3, \mathtt{w}_5 \bullet \\ \quad \texttt{let}\ \mathtt{z}_1, \mathtt{z}_2 \bullet \mathtt{z}_1 > \mathtt{z}_2\ \&\ \texttt{out!}(\mathtt{z}_1 - \mathtt{x}) \to \texttt{Skip} \end{array}\right) \end{array}\right) \end{array}\right)$$

We now have two choices again corresponding to the independent evolutions of

the parallel actions. If the left-hand action evolves first, we have the following.

$\xrightarrow{\mathtt{out!}\mathtt{w}_6}$ [Rules (4), (7), and (15)]

$$\left(\begin{array}{l}\left(\begin{array}{l}\mathtt{w}_0 \in \mathbb{Z} \land \mathtt{w}_1 = 2 \land \mathtt{w}_2 \in \mathbb{Z} \land \mathtt{w}_3 \in \mathbb{Z} \land \\ \mathtt{w}_4 = \mathtt{w}_2 \land \mathtt{w}_5 \in \mathbb{Z} \land \mathtt{w}_4 = \mathtt{w}_5 \land \\ \mathtt{w}_6 = \mathtt{w}_2 - \mathtt{w}_1\end{array}\right) \mid \mathtt{x} := \mathtt{w}_1 \\ \models \\ \left(\begin{array}{l}\mathtt{par}\ \mathtt{x} := \mathtt{w}_1;\ \mathtt{var}\,\mathtt{y} := \mathtt{w}_2 \bullet (\mathtt{let}\ \mathtt{y} \bullet \mathtt{Skip}) \\ \quad [\![\{\!|\, \mathtt{int} \,|\!\}]\!] \\ \left(\begin{array}{l}\mathtt{par}\ \mathtt{x} := \mathtt{w}_1;\ \mathtt{var}\,\mathtt{z}_1, \mathtt{z}_2 := \mathtt{w}_3, \mathtt{w}_5 \bullet \\ \quad \mathtt{let}\ \mathtt{z}_1, \mathtt{z}_2 \bullet \mathtt{z}_1 > \mathtt{z}_2 \,\&\, \mathtt{out!}(\mathtt{z}_1 - \mathtt{x}) \rightarrow \mathtt{Skip}\end{array}\right)\end{array}\right)\end{array}\right)$$

And again we have a choice of the silent evolution of the left-hand action, or the evolution of the second action. Continuing with the evolution of the left-hand action, we proceed with the following sequence of transitions.

$\longrightarrow$ [Rules (8) and (15)]

$$\left(\begin{array}{l}\left(\begin{array}{l}\mathtt{w}_0 \in \mathbb{Z} \land \mathtt{w}_1 = 2 \land \mathtt{w}_2 \in \mathbb{Z} \land \mathtt{w}_3 \in \mathbb{Z} \land \\ \mathtt{w}_4 = \mathtt{w}_2 \land \mathtt{w}_5 \in \mathbb{Z} \land \mathtt{w}_4 = \mathtt{w}_5 \land \\ \mathtt{w}_6 = \mathtt{w}_2 - \mathtt{w}_1\end{array}\right) \mid \mathtt{x} := \mathtt{w}_1 \\ \models \\ \left(\begin{array}{l}\mathtt{par}\ \mathtt{x} := \mathtt{w}_1;\ \mathtt{var}\,\mathtt{y} := \mathtt{w}_2;\ \mathtt{end}\,\mathtt{y} \bullet \mathtt{Skip} \\ \quad [\![\{\!|\, \mathtt{int} \,|\!\}]\!] \\ \left(\begin{array}{l}\mathtt{par}\ \mathtt{x} := \mathtt{w}_1;\ \mathtt{var}\,\mathtt{z}_1, \mathtt{z}_2 := \mathtt{w}_3, \mathtt{w}_5 \bullet \\ \quad \mathtt{let}\ \mathtt{z}_1, \mathtt{z}_2 \bullet \mathtt{z}_1 > \mathtt{z}_2 \,\&\, \mathtt{out!}(\mathtt{z}_1 - \mathtt{x}) \rightarrow \mathtt{Skip}\end{array}\right)\end{array}\right)\end{array}\right)$$

$\longrightarrow$ [Rules (12), (7) and (16)]

$$\left(\begin{array}{l}\left(\begin{array}{l}\mathtt{w}_0 \in \mathbb{Z} \land \mathtt{w}_1 = 2 \land \mathtt{w}_2 \in \mathbb{Z} \land \mathtt{w}_3 \in \mathbb{Z} \land \\ \mathtt{w}_4 = \mathtt{w}_2 \land \mathtt{w}_5 \in \mathbb{Z} \land \mathtt{w}_4 = \mathtt{w}_5 \land \\ \mathtt{w}_6 = \mathtt{w}_2 - \mathtt{w}_1 \land \mathtt{w}_3 > \mathtt{w}_5\end{array}\right) \mid \mathtt{x} := \mathtt{w}_1 \\ \models \\ \left(\begin{array}{l}\mathtt{par}\ \mathtt{x} := \mathtt{w}_1;\ \mathtt{var}\,\mathtt{y} := \mathtt{w}_2;\ \mathtt{end}\,\mathtt{y} \bullet \mathtt{Skip} \\ \quad [\![\{\!|\, \mathtt{int} \,|\!\}]\!] \\ \left(\begin{array}{l}\mathtt{par}\ \mathtt{x} := \mathtt{w}_1;\ \mathtt{var}\,\mathtt{z}_1, \mathtt{z}_2 := \mathtt{w}_3, \mathtt{w}_5 \bullet \\ \quad \mathtt{let}\ \mathtt{z}_1, \mathtt{z}_2 \bullet \mathtt{out!}(\mathtt{z}_1 - \mathtt{x}) \rightarrow \mathtt{Skip}\end{array}\right)\end{array}\right)\end{array}\right)$$

$\xrightarrow{\mathtt{out}.\mathtt{w}_7}$ [Rules (4), (7) and (16)]

$$\left(\begin{array}{l}\left(\begin{array}{l}\mathtt{w}_0 \in \mathbb{Z} \land \mathtt{w}_1 = 2 \land \mathtt{w}_2 \in \mathbb{Z} \land \mathtt{w}_3 \in \mathbb{Z} \land \\ \mathtt{w}_4 = \mathtt{w}_2 \land \mathtt{w}_5 \in \mathbb{Z} \land \mathtt{w}_4 = \mathtt{w}_5 \land \\ \mathtt{w}_6 = \mathtt{w}_2 - \mathtt{w}_1 \land \mathtt{w}_3 > \mathtt{w}_5 \land \mathtt{w}_7 = \mathtt{w}_3 - \mathtt{w}_1\end{array}\right) \mid \mathtt{x} := \mathtt{w}_1 \\ \models \\ \left(\begin{array}{l}\mathtt{par}\ \mathtt{x} := \mathtt{w}_1;\ \mathtt{var}\,\mathtt{y} := \mathtt{w}_2;\ \mathtt{end}\,\mathtt{y} \bullet \mathtt{Skip} \\ \quad [\![\{\!|\, \mathtt{int} \,|\!\}]\!] \\ \mathtt{par}\ \mathtt{x} := \mathtt{w}_1;\ \mathtt{var}\,\mathtt{z}_1, \mathtt{z}_2 := \mathtt{w}_3, \mathtt{w}_5 \bullet (\mathtt{let}\ \mathtt{z}_1, \mathtt{z}_2 \bullet \mathtt{Skip})\end{array}\right)\end{array}\right)$$

$$
\begin{array}{ll}
\longrightarrow & \text{[Rules (8) and (16)]} \\
\left(\begin{array}{l}
\left(\begin{array}{l}
\mathtt{w}_0 \in \mathbb{Z} \land \mathtt{w}_1 = 2 \land \mathtt{w}_2 \in \mathbb{Z} \land \mathtt{w}_3 \in \mathbb{Z} \land \\
\mathtt{w}_4 = \mathtt{w}_2 \land \mathtt{w}_5 \in \mathbb{Z} \land \mathtt{w}_4 = \mathtt{w}_5 \land \\
\mathtt{w}_6 = \mathtt{w}_2 - \mathtt{w}_1 \land \mathtt{w}_3 > \mathtt{w}_5 \land \mathtt{w}_7 = \mathtt{w}_3 - \mathtt{w}_1
\end{array}\right) \mid \mathtt{x} := \mathtt{w}_1 \\
\models \\
\left(\begin{array}{l}
\mathtt{par}\ \mathtt{x} := \mathtt{w}_1;\ \mathtt{var}\ \mathtt{y} := \mathtt{w}_2;\ \mathtt{end}\ \mathtt{y} \bullet \mathtt{Skip} \\
\quad [\![\{\!| \mathtt{int} |\!\}]\!] \\
\mathtt{par}\ \mathtt{x} := \mathtt{w}_1;\ \mathtt{var}\ \mathtt{z}_1, \mathtt{z}_2 := \mathtt{w}_3, \mathtt{w}_5;\ \mathtt{end}\ \mathtt{z}_1, \mathtt{z}_2 \bullet \mathtt{Skip}
\end{array}\right)
\end{array}\right) & \\
\longrightarrow & \text{[Rule (14)]} \\
\left(\begin{array}{l}
\left(\begin{array}{l}
\mathtt{w}_0 \in \mathbb{Z} \land \mathtt{w}_1 = 2 \land \mathtt{w}_2 \in \mathbb{Z} \land \mathtt{w}_3 \in \mathbb{Z} \land \\
\mathtt{w}_4 = \mathtt{w}_2 \land \mathtt{w}_5 \in \mathbb{Z} \land \mathtt{w}_4 = \mathtt{w}_5 \land \\
\mathtt{w}_6 = \mathtt{w}_2 - \mathtt{w}_1 \land \mathtt{w}_3 > \mathtt{w}_5 \land \mathtt{w}_7 = \mathtt{w}_3 - \mathtt{w}_1
\end{array}\right) \mid \mathtt{x} := \mathtt{w}_1 \\
\models \\
\mathtt{Skip}
\end{array}\right) &
\end{array}
$$

Various interleavings of the evolution of each of the parallel actions are possible. The above is just an example. A second option, for instance, carries out all the evolutions of the right-hand side action to `Skip` before evolving the left-hand action. In this case, the order of communication of $y - x$ and $z_1 - x$ on *out* changes. The end configuration, with a careful choice of the names of the symbolic variables as illustrated before, is the same in all cases. □

In the next section, we explain how the operational semantics is used to define tests based on *Circus* models of a system.

2.3 Testing in *Circus*

In previous work, we have instantiated Gaudel's long-standing testing theory to *Circus* [15]. The conformance relation we have considered is process refinement. This is the UTP notion of refinement applied to processes, that is, to their main actions, where the state components are taken as local variables.

As already said, we take the view that, in specifications, divergences are mistakes. In programs, they are observed as deadlocks. We, therefore, consider a theory for divergence-free models and systems under test (SUT). In this case, the refinement relation of *Circus* can be characterised by the conjunction of a traces refinement relation, and a *conf* relation that requires reduction of deadlock. This is proved in [7], where both relations are defined in the UTP *Circus* theory.

Accordingly, we have defined separate exhaustive test sets for traces refinement and *conf*. We have taken advantage of the symbolic nature of the *Circus* operational semantics, and defined the tests symbolically. These definitions specify how concrete tests can be obtained by a process of instantiation.

A test for traces refinement is constructed by considering a trace of the *Circus* model and one of the events that cannot be used to extend that trace to

obtain a new trace of the *Circus* model [5]. Such events are called the forbidden continuations of the trace. Traces and forbidden continuations are characterised symbolically. The exhaustive test set includes all the tests formed by considering all the traces and all their forbidden continuations.

For the process PA in Example 2, we have traces of communications over $inpA$, $inpB$, and int. Below, we present a symbolic trace that specifies some of them; it has an associated constraint over the symbolic variables used in the specification of the trace. We call these pairs constrained symbolic traces.

$$(\langle \mathtt{inpA}.\mathtt{w}_2, \mathtt{inpB}.\mathtt{w}_3, \mathtt{int}.\mathtt{w}_5 \rangle, \mathtt{w}_2 \in \mathbb{Z} \land \mathtt{w}_3 \in \mathbb{Z} \land \mathtt{w}_5 = \mathtt{w}_2)$$

Roughly speaking, the constrained symbolic trace can be obtained by evaluating the operational semantics, collecting the labels together, and keeping the constraint over the symbolic variables used in the labels. The ? and ! decorations that determine whether the communications are inputs or outputs are ignored.

There is a forbidden continuation of this trace for each of the channels in scope. The only possible continuations involve communications over out, but not all of them are allowed. For example, the following is the specification of the forbidden continuations involving $inpA$; it is a constrained symbolic event.

$$(\mathtt{inpA}.\mathtt{w}_6, \mathtt{w}_2 \in \mathbb{Z} \land \mathtt{w}_3 \in \mathbb{Z} \land \mathtt{w}_5 = \mathtt{w}_2)$$

It records the constraint of the trace, and imposes no restriction on the value w_6 communicated via $inpA$, since no value is allowed. The specification for the forbidden continuations involving out, on the other hand, is as follows.

$$(\mathtt{out}.\mathtt{w}_6, \mathtt{w}_2 \in \mathbb{Z} \land \mathtt{w}_3 \in \mathbb{Z} \land \mathtt{w}_5 = \mathtt{w}_2 \land \mathtt{w}_6 \neq \mathtt{w}_2 - 2 \land \mathtt{w}_6 \neq \mathtt{w}_3 - 2)$$

The symbolic tests corresponding to the above trace and the forbidden continuation above (involving out) is as follows.

$$\begin{array}{l} inc \rightarrow inpA?w_2 : w_2 \in \mathbb{Z} \rightarrow \\ inc \rightarrow inpB?w_3 : w_2 \in \mathbb{Z} \land w_3 \in \mathbb{Z} \rightarrow \\ inc \rightarrow int?w_5 : w_2 \in \mathbb{Z} \land w_3 \in \mathbb{Z} \land w_5 = w_2 \rightarrow pass \rightarrow \\ out?w_6 : w_2 \in \mathbb{Z} \land w_3 \in \mathbb{Z} \land w_5 = w_2 \land w_6 \neq w_2 - 2 \land w_6 \neq w3 - 2 \rightarrow \\ fail \rightarrow \mathbf{Stop} \end{array}$$

We use extra special events inc, $pass$ and $fail$ to indicate a verdict. In the execution of a testing experiment, the test is run in parallel with the SUT, with all the model events hidden, so that the interaction between the test and the SUT cannot be affected by the environment. In our example, the communications over $inpA$, $inpB$, and int are hidden. The last special event observed in a testing experiment provides the verdict. Due the possibility of nondeterminism, a trace of the model is not necessarily available in the SUT. The inc event indicates an inconclusive verdict: the SUT has not performed the proposed trace. If it does perform the trace, we have a $pass$ event, but if the SUT proceeds to engage in the forbidden communication, then we have a $fail$.

The last event is that observed before the testing experiment leads to a deadlock. As already hinted, we do assume that we can observe a deadlock. In practice, this requires the definition of a timeout.

A possible concrete test satisfying the test specification above is as follows.

$$inc \rightarrow inpA.0 \rightarrow inc \rightarrow inpB.1 \rightarrow inc \rightarrow int.0 \rightarrow pass \rightarrow out.2 \rightarrow fail \rightarrow \mathbf{Stop}$$

There are, of course, infinitely many other choices, as there may be infinitely many test specifications, in the case, for example, of nonterminating processes.

In the tests for *conf*, we use the traces and acceptances of a process. In the exhaustive test set for *conf*, we have all tests formed by considering all traces of the model, and all the acceptance sets after each of them.

In a *conf* test we check that, after the trace, the SUT does not deadlock if it is offered all the events of an acceptance set. Acceptance sets, like refusals, are more interesting for nondeterministic processes. So, we consider the action E in Example 1. After the empty trace $(\langle\rangle, \texttt{True})$, the specification of the minimal sets of acceptances is the following set of constrained symbolic events. They record whether the communications are inputs or outputs. As explained below, this is important in the creation of the concrete acceptance sets and *conf* tests.

$$\{(\texttt{out!}\mathtt{w_3}, \mathtt{w_3} > 0), (\texttt{inp?}\mathtt{w_3}, \mathtt{w_3} \in \mathbb{Z})\}$$

Roughly speaking, this is obtained by picking one of the continuations from each of the stable states that can be reached via the empty trace. Stable states are those from which there is no silent transition available. In our example, the stable states are those from which just a transition with label $\texttt{out!}\mathtt{w_3}$ or label $\texttt{inp?}\mathtt{w_3}$ is available. These labels define the continuations.

The symbolic test for *conf* defined by the empty trace and the constrained symbolic acceptance set above is as follows.

$$fail \rightarrow (out!w_3 : w_3 > 0 \rightarrow pass \rightarrow \mathbf{Stop} \,\Box\, inp?w_3 : w_3 \in \mathbb{Z} \rightarrow pass \rightarrow \mathbf{Stop})$$

Since the trace is empty, there is no need for *inc* events. Before offering the SUT all the events of the acceptance set, we have a *fail*. The SUT cannot deadlock when all events of an acceptance are available, so if it accepts any of them, then we have a *pass*. Otherwise, the *fail* verdict stands.

In instantiating the above test, we can obtain the concrete test below.

$$fail \rightarrow (out?w_3 : w_3 > 0 \rightarrow pass \rightarrow \mathbf{Stop} \,\Box\, inp.0 \rightarrow pass \rightarrow \mathbf{Stop})$$

The output in the model becomes an input in the test, since any output produced by the SUT is acceptable as long as it satisfies the associated constraint. For the input, a concrete test chooses a particular value satisfying the constraint.

The constraints in the symbolic tests for both traces refinement and *conf* define the constraint-satisfaction problems that need to be solved to obtain concrete tests. They provide a concise account of the state operations and their properties. Selection of concrete tests can use criteria based on coverage of the

symbolic transition system, for instance. In addition, we can use the constraints to apply standard techniques based on uniformity subdomains. A very simple approach, for instance, considers, to start with, just one concrete test for each symbolic test (so that the constraints are themselves taken as definitions of uniformity subdomains: sets of tests that provide the same verdict).

What the symbolic tests do not provide is support for criteria based on the structure of the models. For example, in the tests above, we have no record of the way in which the variables x, y, z and so on are used. For larger examples, uses of data operations can also be of interest. For instance, the symbolic tests for the *ResourceManager* presented in Section 2.1 do not keep a record of the use of the operations *Cache*, *Insert*, and so on. It is to address this issue that we define a new transition system for *Circus* in the next section.

3 Specification-oriented transition system

The main distinguishing feature of the new transition system is its labels. They record not only events, like in the operational semantics, but also guards and state changes. Additionally, they are expressed in terms of the expressions of the *Circus* model, rather than symbolic variables. For example, for the action E in Example 1, we have transitions with labels $\mathtt{x} := 2$, $\mathtt{y} > \mathtt{z}$, and $\mathtt{out!z}$.

Furthermore, the specification-oriented system has no silent transitions; they correspond to evolutions that are not guarded, and do not entail any communication or state change. These transitions do not capture observable behaviour, and so are not interesting from a testing point of view.

We first discuss the definition of the new transition system for processes (Section 3.1). It is specified in terms of a transition relation for actions (Section 3.4), which is itself defined in terms of two other relations (Sections 3.2 and 3.3).

3.1 Processes

Like in the operational semantics, we have a transition relation $\Longrightarrow$ between texts of process. It is defined in terms of the corresponding relation for actions by the transition rule below. The labels are triples including a guard, an event, and an action. As mentioned above, there are no silent transitions.

We have a single transition rule, which allows us to lift transitions of the main action of a process in its local state to the process itself.

$$\frac{(state(\mathtt{P}_1) \models maction(\mathtt{P}_1)) \stackrel{1}{\Longrightarrow} (state(\mathtt{P}_2) \models maction(\mathtt{P}_2))}{\mathtt{P}_1 \stackrel{1}{\Longrightarrow} \mathtt{P}_2} \tag{1}$$

The local state of a process is characterised by the syntactic function *state* (P). It is defined below for basic processes: those originally in the *Circus* notation, and the extended form of process with a `loc` clause used in the operational semantics.

$$state(\mathtt{begin\ state}\ [\mathtt{x} : \mathtt{T}] \bullet \mathtt{A}\ \mathtt{end}) = (\mathtt{w}_0 \in \mathtt{T} \mid \mathtt{x} := \mathtt{w}_0)$$
$$state(\mathtt{begin\ state}\ [\mathtt{x} : \mathtt{T}]\ \mathtt{loc}\ (\mathtt{c} \mid \mathtt{s}) \bullet \mathtt{A}\ \mathtt{end}) = (\mathtt{c} \mid \mathtt{s})$$

where $\mathtt{w}_0$ is a fresh symbolic variable. As mentioned in Section 2.2, there is no

need to consider the composed processes, which are defined in terms of basic processes. Another syntactic function *maction* extracts the main action of a basic process. Its simple definition is omitted.

Unlike in the operational semantics, we do not have a rule to introduce the extended form of basic process as a first step of the evaluation. That is a silent transition, which we do not keep in the specification-oriented system.

For actions, a transition $(\mathtt{c_1} \mid \mathtt{s_1} \models \mathtt{A_1}) \overset{(\mathtt{g},\mathtt{e},\mathtt{A})}{\Longrightarrow} (\mathtt{c_2} \mid \mathtt{s_2} \models \mathtt{A_2})$ establishes that in the state characterised by $(c_1 \mid s_1)$, if the guard g holds, then in the execution of A_1 the event e takes place, and afterwards A is executed. The new state is then characterised by $(c_2 \mid s_2)$ and the remaining action to execute is A_2. In the label, if the guard is `True`, we can omit it, and write just `(e,A)`. Similarly, we omit the event if its value is ϵ, and the action, if omitted, is `Skip`. We do not have silent transitions, here defined as transitions with label $(\mathtt{True}, \epsilon, \mathtt{Skip})$. So, at least one of the components of a label has to be given explicitly. If it has only one component given explicitly, we do not use the tuple notation.

The language used to write guards, events, and actions is *Circus* [23]. For actions, however, we include the extensions necessary to express the operational semantics, add the UTP constructs for variable declaration ($\mathbf{var}\ x : T$) and undeclaration ($\mathbf{end}\ x$), and two new constructs for parallelism and external choice.

The definition of $(\mathtt{c_1} \mid \mathtt{s_1} \models \mathtt{A_1}) \overset{(\mathtt{g},\mathtt{e},\mathtt{A})}{\Longrightarrow} (\mathtt{c_2} \mid \mathtt{s_2} \models \mathtt{A_2})$ uses a succession of other transition relations that we define in the next sections.

3.2 Specification labels

The first relation $(\mathtt{c_1} \mid \mathtt{s_1} \models \mathtt{A_1}) \overset{(\mathtt{g},\mathtt{e},\mathtt{A})}{\Longrightarrow}_L (\mathtt{c_2} \mid \mathtt{s_2} \models \mathtt{A_2})$ associates configurations that are already related by a transition of the operational semantics. It, however, records more information in the labels, as explained above, and formalised below by the transition rules that define this new relation.

Basic actions There are three rules for basic actions presented below: one for designs, one for schemas, and one for assignment. They are basically the same as the corresponding rules of the operational semantics (see Appendix A). The difference is that we record the executed action (state change) in the label.

$$\frac{c \wedge (s;\ p) \wedge (\exists\, v' \bullet s;\ Q)}{(\mathtt{c} \mid \mathtt{s} \models \mathtt{p} \vdash \mathtt{Q}) \overset{\mathtt{p} \vdash \mathtt{Q}}{\Longrightarrow}_L (\mathtt{c} \wedge (\mathtt{s};\ \mathtt{Q}\,[\mathtt{w_0}/\mathtt{v'}]) \mid \mathtt{s};\ \mathtt{v} := \mathtt{w_0} \models \mathtt{Skip})} \quad v' = out\alpha s \qquad (2)$$

$$\frac{c \wedge (s;\ \mathbf{pre}\ Op)}{(\mathtt{c} \mid \mathtt{s} \models \mathtt{Op}) \overset{\mathtt{Op}}{\Longrightarrow} (\mathtt{c} \wedge (\mathtt{s};\ \mathtt{Op}\,[\mathtt{w_0}/\mathtt{v'}]) \mid \mathtt{s};\ \mathtt{v} := \mathtt{w_0} \models \mathtt{Skip})} \quad v' = out\alpha s \qquad (3)$$

$$\frac{c}{(\mathtt{c} \mid \mathtt{s} \models \mathtt{v} := \mathtt{e}) \overset{\mathtt{v}:=\mathtt{e}}{\Longrightarrow}_L (\mathtt{c} \wedge (\mathtt{s};\ \mathtt{w}_0 = \mathtt{e}) \mid \mathtt{s};\ \mathtt{v} := \mathtt{w}_0 \models \mathtt{Skip})} \tag{4}$$

What we do not have are rules corresponding to those in the operational semantics that cover the situation where the precondition of a design or schema is false, and the action diverges. These are not useful in our work on testing, where, as previously explained, we assume the absence of divergence.

Example 3. For components of our example action E introduced in Example 1, we have the two transitions below. The numbers refer to the transition rules of the new relation presented above, rather than those of the operational semantics.

$$(\mathtt{w}_0 \in \mathbb{Z} \wedge \mathtt{w}_1 \in \mathbb{Z} \mid \mathtt{x}, \mathtt{y} := \mathtt{w}_0, \mathtt{w}_1 \models \mathtt{x} := 2)$$

$$\overset{\mathtt{x}:=2}{\Longrightarrow}_L \qquad \text{[Rule (4)]}$$

$$(\mathtt{w}_0 \in \mathbb{Z} \wedge \mathtt{w}_1 \in \mathbb{Z} \wedge \mathtt{w}_2 = 2 \mid \mathtt{x}, \mathtt{y} := \mathtt{w}_2, \mathtt{w}_1 \models \mathtt{Skip})$$

$$\left(\begin{array}{l} \mathtt{w}_0 \in \mathbb{Z} \wedge \mathtt{w}_1 \in \mathbb{Z} \wedge \mathtt{w}_2 = 2 \wedge \mathtt{w}_1 > \mathtt{w}_2 \wedge \mathtt{w}_3 = \mathtt{w}_1 - \mathtt{w}_2 \mid \mathtt{x}, \mathtt{y} := \mathtt{w}_2, \mathtt{w}_1 \\ \models \\ \mathtt{x} := \mathtt{y} \end{array}\right)$$

$$\overset{\mathtt{x}:=\mathtt{y}}{\Longrightarrow}_L \qquad \text{[Rule (4)]}$$

$$\left(\begin{array}{l} \mathtt{w}_0 \in \mathbb{Z} \wedge \mathtt{w}_1 \in \mathbb{Z} \wedge \mathtt{w}_2 = 2 \wedge \mathtt{w}_1 > \mathtt{w}_2 \wedge \mathtt{w}_3 = \mathtt{w}_1 - \mathtt{w}_2 \wedge \mathtt{w}_4 = \mathtt{w}_2 \mid \mathtt{x}, \mathtt{y} := \mathtt{w}_4, \mathtt{w}_1 \\ \models \\ \mathtt{Skip} \end{array}\right)$$

□

We do not use symbolic variables in labels. We still, however, keep the characterisation of the state in terms of symbolic variables. This allows the combined use of the operational semantics and the specification-oriented transition system. This is useful both in the definition of the new transition system, and in the testing techniques that we plan to explore, since as explained in Section 2.3, *Circus* tests are expressed in terms of the symbolic variables.

Guards and prefixings As previously mentioned, guards are also recorded in labels. We present below a rule similar to Rule (12) of the operational semantics, but which records the guard in the label.

$$\frac{c \wedge (s;\ g)}{(\mathtt{c} \mid \mathtt{s} \models \mathtt{g} \,\&\, \mathtt{A}) \overset{\mathtt{g}}{\Longrightarrow}_L (\mathtt{c} \wedge (\mathtt{s};\ \mathtt{g}) \mid \mathtt{s} \models \mathtt{A})} \tag{5}$$

There is no rule here and in the operational semantics for when the guard does

not hold. This is a deadlock, represented by the absence of available transitions.

Example 4.

$$\left(\begin{array}{l} \mathtt{w_0} \in \mathbb{Z} \land \mathtt{w_1} \in \mathbb{Z} \land \mathtt{w_2} = 2 \mid \mathtt{x}, \mathtt{y} := \mathtt{w_2}, \mathtt{w_1} \\ \models \\ \mathtt{y} > \mathtt{x} \,\&\, \mathtt{out!}(\mathtt{y} - \mathtt{x}) \rightarrow \mathtt{Skip} \end{array}\right)$$
$$\stackrel{\mathtt{y}>\mathtt{x}}{\Longrightarrow}_L \qquad \text{[Rule (5)]}$$
$$\left(\begin{array}{l} \mathtt{w_0} \in \mathbb{Z} \land \mathtt{w_1} \in \mathbb{Z} \land \mathtt{w_2} = 2 \land \mathtt{w_1} > \mathtt{w_2} \mid \mathtt{x}, \mathtt{y} := \mathtt{w_2}, \mathtt{w_1} \\ \models \\ \mathtt{out!}(\mathtt{y} - \mathtt{x}) \rightarrow \mathtt{Skip} \end{array}\right)$$

□

As opposed to the transitions in the operational semantics, here the labels for the transitions that apply to output prefixings record the expressions $\mathtt{e}$ whose values are output (rather than symbolic variables representing those values).

$$\frac{c}{(\mathtt{c} \mid \mathtt{s} \models \mathtt{d!e} \rightarrow \mathtt{A}) \stackrel{\mathtt{d!e}}{\Longrightarrow}_L (\mathtt{c} \land (\mathtt{s};\ \mathtt{w_0} = \mathtt{e}) \mid \mathtt{s} \models \mathtt{A})} \tag{6}$$

As discussed before, in the labels of $\Longrightarrow_L$, there is no use of symbolic variables. These labels record the text of the specification, rather than events with evaluated values (represented by symbolic variables). In this way, they record, for instance, the specification variables used in the communication $\mathtt{d!e}$.

Example 5.

$$\left(\begin{array}{l} \mathtt{w_0} \in \mathbb{Z} \land \mathtt{w_1} \in \mathbb{Z} \land \mathtt{w_2} = 2 \land \mathtt{w_1} > \mathtt{w_2} \mid \mathtt{x}, \mathtt{y} := \mathtt{w_2}, \mathtt{w_1} \\ \models \\ \mathtt{out!}(\mathtt{y} - \mathtt{x}) \rightarrow \mathtt{Skip} \end{array}\right)$$
$$\stackrel{\mathtt{out.}(\mathtt{y}-\mathtt{x})}{\Longrightarrow}_L \qquad \text{[Rule (6)]}$$
$$\left(\begin{array}{l} \mathtt{w_0} \in \mathbb{Z} \land \mathtt{w_1} \in \mathbb{Z} \land \mathtt{w_2} = 2 \land \mathtt{w_1} > \mathtt{w_2} \land \mathtt{w_3} = \mathtt{w_1} - \mathtt{w_2} \mid \mathtt{x}, \mathtt{y} := \mathtt{w_2}, \mathtt{w_1} \\ \models \\ \mathtt{Skip} \end{array}\right)$$

□

In the case of an input, the label records the input variable.

$$\frac{c \land T \neq \varnothing \qquad x \notin \alpha s}{(\mathtt{c} \mid \mathtt{s} \models \mathtt{d?x} : \mathtt{T} \rightarrow \mathtt{A}) \stackrel{\mathtt{d?x}}{\Longrightarrow} (\mathtt{c} \land \mathtt{w_0} \in \mathtt{T} \mid \mathtt{s};\ \mathtt{var\,x} := \mathtt{w_0} \models \mathtt{let\ x} \bullet \mathtt{A})} \tag{7}$$

Just like in CSP, the input implicitly declares the input variable $\mathtt{x}$.

Example 6.

$$\left(\begin{array}{l}\mathtt{w}_0 \in \mathbb{Z} \land \mathtt{w}_1 \in \mathbb{Z} \land \mathtt{w}_2 = 2 \mid \mathtt{x}, \mathtt{y} := \mathtt{w}_2, \mathtt{w}_1 \\ \models \\ \mathtt{inp?z} \to \mathtt{Stop}\end{array}\right)$$

$$\overset{\mathtt{inp?z}}{\Longrightarrow}_L \qquad \text{[Rules (5) and (9)]}$$

$$\left(\begin{array}{l}\mathtt{w}_0 \in \mathbb{Z} \land \mathtt{w}_1 \in \mathbb{Z} \land \mathtt{w}_2 = 2 \land \mathtt{w}_3 \in \mathbb{Z} \mid \mathtt{x}, \mathtt{y} := \mathtt{w}_2, \mathtt{w}_1;\ \mathtt{var}\,\mathtt{z} := \mathtt{w}_3 \\ \models \\ \mathtt{let}\ \mathtt{z} \bullet \mathtt{Stop}\end{array}\right)$$

□

Variables To record a variable declaration in a label, we use the UTP variable declaration construct **var** x. It is not available in *Circus*, originally, but can be defined as $\exists\, x \bullet$ **Skip** in the UTP theory for *Circus*.

$$\frac{c \land T \neq \varnothing \qquad x \notin \alpha s}{(\mathtt{c} \mid \mathtt{s} \models \mathtt{var}\,\mathtt{x} : \mathtt{T} \bullet \mathtt{A}) \overset{(\mathtt{var}\ \mathtt{x}:\mathtt{T})}{\Longrightarrow}_L (\mathtt{c} \land \mathtt{w}_0 \in \mathtt{T} \mid \mathtt{s};\ \mathtt{var}\,\mathtt{x} := \mathtt{w}_0 \models \mathtt{let}\ \mathtt{x} \bullet \mathtt{A})} \tag{8}$$

Like in the operational semantics, we assume that variable names are not reused.

When the action in the scope of a variable declaration finishes, the end of the scope is also recorded. For that we use the UTP undeclaration construct **end** x.

$$\frac{c}{(\mathtt{c} \mid \mathtt{s} \models \mathtt{let}\ \mathtt{x} \bullet \mathtt{Skip}) \overset{(\mathtt{end}\ \mathtt{x})}{\Longrightarrow}_L (\mathtt{c} \mid \mathtt{s};\ \mathtt{end}\,\mathtt{x} \models \mathtt{Skip})} \tag{9}$$

Action operators There are standard rules that reflect the fact that evolution of a component action leads to the evolution of the composed action that uses it.

$$\frac{(\mathtt{c}_1 \mid \mathtt{s}_1 \models \mathtt{A}_1) \overset{l}{\Longrightarrow}_L (\mathtt{c}_2 \mid \mathtt{s}_2 \models \mathtt{A}_2)}{(\mathtt{c}_1 \mid \mathtt{s}_1 \models \mathtt{let}\ \mathtt{x} \bullet \mathtt{A}_1) \overset{l}{\Longrightarrow}_L (\mathtt{c}_2 \mid \mathtt{s}_2 \models \mathtt{let}\ \mathtt{x} \bullet \mathtt{A}_2)} \tag{10}$$

$$\frac{(\mathtt{c}_1 \mid \mathtt{s}_1 \models \mathtt{A}_1) \overset{l}{\Longrightarrow}_L (\mathtt{c}_2 \mid \mathtt{s}_2 \models \mathtt{A}_2)}{(\mathtt{c}_1 \mid \mathtt{s}_1 \models \mathtt{A}_1;\ \mathtt{B}) \overset{l}{\Longrightarrow}_L (\mathtt{c}_2 \mid \mathtt{s}_2 \models \mathtt{A}_2;\ \mathtt{B})} \tag{11}$$

For the silent transitions of the operational semantics that are involved in the evolution of a composed action, we have no corresponding transition rule. For instance, we have no specific rule for an action $\mathtt{Skip};\ \mathtt{A}$ or an action $\mathtt{A}_1 \sqcap \mathtt{A}_2$. In the following section, we give a transition relation that handles these actions.

Example 7. We consider again the action E of Example 1, and present below transitions that are justified by the rules for the specification-oriented relation. We observe that, in many cases, no such rule applies, and we indicate again the transitions of the operational semantics that are possible.

$$(\mathtt{w_0} \in \mathbb{Z} \wedge \mathtt{w_1} \in \mathbb{Z} \mid \mathtt{x}, \mathtt{y} := \mathtt{w_0}, \mathtt{w_1} \models \mathtt{E})$$

$$\overset{\mathtt{x:=2}}{\Longrightarrow}_L \qquad \text{[Rules (4) and (11)]}$$

$$\left(\begin{array}{l} \mathtt{w_0} \in \mathbb{Z} \wedge \mathtt{w_1} \in \mathbb{Z} \wedge \mathtt{w_2} = 2 \mid \mathtt{x}, \mathtt{y} := \mathtt{w_2}, \mathtt{w_1} \\ \models \\ \mathtt{Skip};\ (\mathtt{y} > \mathtt{x} \mathbin{\&} \mathtt{out!}(\mathtt{y} - \mathtt{x}) \rightarrow \mathtt{Skip} \sqcap \mathtt{inp?z} \rightarrow \mathtt{Stop});\ \mathtt{x} := \mathtt{y} \end{array}\right)$$

$$\longrightarrow$$

$$\left(\begin{array}{l} \mathtt{w_0} \in \mathbb{Z} \wedge \mathtt{w_1} \in \mathbb{Z} \wedge \mathtt{w_2} = 2 \mid \mathtt{x}, \mathtt{y} := \mathtt{w_2}, \mathtt{w_1} \\ \models \\ (\mathtt{y} > \mathtt{x} \mathbin{\&} \mathtt{out!}(\mathtt{y} - \mathtt{x}) \rightarrow \mathtt{Skip} \sqcap \mathtt{inp?z} \rightarrow \mathtt{Stop});\ \mathtt{x} := \mathtt{y} \end{array}\right)$$

As before, we consider each of the options of the internal choice in turn.

$$\longrightarrow$$

$$\left(\begin{array}{l} \mathtt{w_0} \in \mathbb{Z} \wedge \mathtt{w_1} \in \mathbb{Z} \wedge \mathtt{w_2} = 2 \mid \mathtt{x}, \mathtt{y} := \mathtt{w_2}, \mathtt{w_1} \\ \models \\ (\mathtt{y} > \mathtt{x} \mathbin{\&} \mathtt{out!}(\mathtt{y} - \mathtt{x}) \rightarrow \mathtt{Skip});\ \mathtt{x} := \mathtt{y} \end{array}\right)$$

$$\overset{\mathtt{y>x}}{\Longrightarrow}_L \qquad \text{[Rules (5) and (11)]}$$

$$\left(\begin{array}{l} \mathtt{w_0} \in \mathbb{Z} \wedge \mathtt{w_1} \in \mathbb{Z} \wedge \mathtt{w_2} = 2 \wedge \mathtt{w_1} > \mathtt{w_2} \mid \mathtt{x}, \mathtt{y} := \mathtt{w_2}, \mathtt{w_1} \\ \models \\ (\mathtt{out!}(\mathtt{y} - \mathtt{x}) \rightarrow \mathtt{Skip});\ \mathtt{x} := \mathtt{y} \end{array}\right)$$

$$\overset{\mathtt{out!(y-x)}}{\Longrightarrow}_L \qquad \text{[Rules (6) and (11)]}$$

$$\left(\begin{array}{l} \mathtt{w_0} \in \mathbb{Z} \wedge \mathtt{w_1} \in \mathbb{Z} \wedge \mathtt{w_2} = 2 \wedge \mathtt{w_1} > \mathtt{w_2} \wedge \mathtt{w_3} = \mathtt{w_1} - \mathtt{w_2} \mid \mathtt{x}, \mathtt{y} := \mathtt{w_2}, \mathtt{w_1} \\ \models \\ \mathtt{Skip};\ \mathtt{x} := \mathtt{y} \end{array}\right)$$

$$\longrightarrow$$

$$\left(\begin{array}{l} \mathtt{w_0} \in \mathbb{Z} \wedge \mathtt{w_1} \in \mathbb{Z} \wedge \mathtt{w_2} = 2 \wedge \mathtt{w_1} > \mathtt{w_2} \wedge \mathtt{w_3} = \mathtt{w_1} - \mathtt{w_2} \mid \mathtt{x}, \mathtt{y} := \mathtt{w_2}, \mathtt{w_1} \\ \models \\ \mathtt{x} := \mathtt{y} \end{array}\right)$$

$$\overset{\mathtt{x:=y}}{\Longrightarrow}_L \qquad \text{[Rule (2)]}$$

$$\left(\begin{array}{l} \mathtt{w_0} \in \mathbb{Z} \wedge \mathtt{w_1} \in \mathbb{Z} \wedge \mathtt{w_2} = 2 \wedge \mathtt{w_1} > \mathtt{w_2} \wedge \mathtt{w_3} = \mathtt{w_1} - \mathtt{w_2} \wedge \mathtt{w_4} = \mathtt{w_2} \mid \mathtt{x}, \mathtt{y} := \mathtt{w_4}, \mathtt{w_1} \\ \models \\ \mathtt{Skip} \end{array}\right)$$

For the second option of the internal choice, we proceed as follows.

$$\longrightarrow$$

$$\left(\begin{array}{l}\mathtt{w_0} \in \mathbb{Z} \land \mathtt{w_1} \in \mathbb{Z} \land \mathtt{w_2} = \mathtt{2} \mid \mathtt{x}, \mathtt{y} := \mathtt{w_2}, \mathtt{w_1} \\ \models \\ (\mathtt{inp?z} \rightarrow \mathtt{Stop});\ \mathtt{x} := \mathtt{y}\end{array}\right)$$

$$\overset{\mathtt{inp?z}}{\Longrightarrow}_L \qquad \text{[Rules (7) and (11)]}$$

$$\left(\begin{array}{l}\mathtt{w_0} \in \mathbb{Z} \land \mathtt{w_1} \in \mathbb{Z} \land \mathtt{w_2} = \mathtt{2} \land \mathtt{w_3} \in \mathbb{Z} \mid \mathtt{x}, \mathtt{y} := \mathtt{w_2}, \mathtt{w_1};\ \mathtt{var}\,\mathtt{z} := \mathtt{w_3} \\ \models \\ (\mathtt{let}\ \mathtt{z} \bullet \mathtt{Stop});\ \mathtt{x} := \mathtt{y}\end{array}\right)$$

From here, we cannot proceed with either kind of transition. □

Parallelism and external choice For these, the use of global variables raises an important issue. As already explained, parallel actions have access to the values of the global variables before the start of the parallelism, and can change their values locally. The name partitions define the updates that become visible after the parallelism finishes. This raises an issue concerning the interpretation of labels of transitions that reflect the evolution of parallel actions.

Example 8. We consider the following action involving an interleaving. (In both the operational and denotational semantics, interleaving is treated as a parallelism with an empty set of synchronisation channels.)

$$PA \mathrel{\widehat{=}} x := 2;\ ((x := 3;\ out!x \rightarrow \mathbf{Skip}) \; [\![\{x\} \mid \{\,\}]\!] \; (x := 4;\ out!x \rightarrow \mathbf{Skip}))$$

A naive approach to recording the evolution of the parallelism could lead to a sequence of labels like $(\mathtt{x} := \mathtt{2}), (\mathtt{x} := \mathtt{3}), (\mathtt{x} := \mathtt{4}), \mathtt{out!x}$. A perhaps reasonable interpretation of this path of execution would then be that the value 4 is output via *out*. This is, however, not necessarily the case, since, if the output comes from the first parallel action, then the value output is 3, and after two outputs, the new value of x is 3, in spite of the intermediate $\mathtt{x} := \mathtt{4}$. The sequence of labels is not an accurate description of a path of execution of *PA*. □

Example 9. A similar situation arises with the external choice below.

$$ECA \mathrel{\widehat{=}} x := 2;\ ((x := 3;\ outA!x \rightarrow \mathbf{Skip}) \,\Box\, (outB!x \rightarrow \mathbf{Skip}))$$

A sequence of labels $(\mathtt{x} := \mathtt{2}), (\mathtt{x} := \mathtt{3});\ \mathtt{outB!x}$ would be misleading, because the assignment $x := 3$ is discarded once the choice for the other action is taken. □

In both cases, the difficulty is related to the fact that labels expose expressions that occur in the local scope of alternative paths of execution. In the operational semantics, this has been addressed by recording in the symbolic variables the evaluated values of the expressions, and using the symbolic variables, rather than the expressions themselves, to write the labels. Here, to record information about the structure of the specification, we need to keep the expressions.

What we need is to record is the use of global variables as local variables in parallel actions and external choices. For *PA* in Example 8, for instance, we need local versions x_l and x_r of x for the left-hand and the right-hand parallel actions. They are declared at the start of the parallelism, and the parallel actions use the local instead of the global variables. When the parallelism terminates, the global variables are updated in accordance with the name sets, and the local variables undeclared. A possible path for the action *PA*, for instance, is as follows.

$(\mathtt{x} := 2), (\mathtt{var}\, \mathtt{x_l}, \mathtt{x_r} := \mathtt{x}, \mathtt{x}),$
$(\mathtt{x_l} = 3), (\mathtt{x_r} := 4), (\mathtt{out!x_l}), (\mathtt{out!x_r}),$
$(\mathtt{x} := \mathtt{x_l};\ \mathtt{end}\, \mathtt{x_l}, \mathtt{x_r})$

A related problem arises from the use of local variables in parallel paths of execution. This is illustrated and explained in the example below.

Example 10. We consider the interleaving and external choice below.

$$PALV \mathrel{\widehat=} x := 2;\ (int?z \rightarrow out!z \rightarrow \mathbf{Skip}) \,|||\, (out!x \rightarrow \mathbf{Skip})$$
$$ECALV \mathrel{\widehat=} x := 2;\ ((\mathtt{var}\, z \bullet x := z;\ outA \rightarrow \mathbf{Skip}) \,\Box\, (outB \rightarrow \mathbf{Skip}))$$

In the case of *PALV*, the labels `int?z` and `out!z` refer to a variable z that is not in the scope of (the state of) *PALV*, but in the local state of its first parallel action. The same holds for the labels `var z` and `x := z` in the case of *ECALV*, which correspond to state changes that are local to the first action in the choice, and that are discarded if an interaction on *outB* occurs. □

In the operational semantics, this is again addressed by recording the local state of parallel actions and of branches of an external choice. The local state is used to evaluate any predicates or expressions when defining the constraint on symbolic variables. Since only the symbolic variables are used in labels, their interpretation is clear. Once again, however, here we need to keep the expressions.

For that, we in fact consider a single global scope declaring all variables. The structure of the specification itself, and the fact that names are not reused, enforces the appropriate use of the variables. In the case, of *PALV*, for instance, we have the following possible sequence of labels $(\mathtt{x} := 2)$, $(\mathtt{varx_l}, \mathtt{x_r} := \mathtt{x}, \mathtt{x})$, `int?z`, $\mathtt{out!x_r}$, `out!z`, $(\mathtt{end}\, \mathtt{z})$, $(\mathtt{x} := \mathtt{x_r};\ \mathtt{end}\, \mathtt{x_l}, \mathtt{x_r})$. In this case, the scope of `z` is declared inside that of the local versions $\mathtt{x_l}$ and $\mathtt{x_r}$ of `x`.

In summary, we provide an alternative view of the parallelism. It no longer creates two local states as in the operational semantics. Instead, the parallelism gives rise to two local copies of the global variables, which coexist, and at the end of the parallelism are used to update the global variables. This is in contrast with the parallel by merge in the UTP, where the the parallel actions work on local copies of the global state, whose variables are undeclared, and the local states are reconciled when needed. This is the view adopted in the *Circus* denotational and operational semantics. Here, we keep an extended global state containing the original (global) variables and their local copies.

To provide a transition system with these characteristics, we use a new construct $\mathtt{spar}\ v \mid v_1 \mid v_2 \mid x := x_1 \bullet A$, which is used to represent a parallel action

A in a state containing the original global variables v, copies v_1 of these variables that are only used by A, and copies v_2 of these variables that are not used by A. In addition, A has write control over the global variables x, which correspond to the variables x_1. In out example action $PALV$, for instance, for the first action v is x, v_1 is x_l, v_2 is x_r, and if we assume that this is the action that has write control over x, then x is itself the programming variable x, and x_1 is x_l.

The transition rule that introduces the use of this new construct is as follows. The label in this case records the declaration of the new variables.

$$\frac{c}{\begin{array}{l}(\mathtt{c} \mid \mathtt{s} \models \mathtt{A_1} \llbracket \mathtt{x1} \mid \mathtt{cs} \mid \mathtt{x2} \rrbracket \mathtt{A_2}) \\ \overset{\mathtt{var}\ \mathtt{v_1,v_r := v,v}}{\Longrightarrow_L} \\ \left(\begin{array}{l}\mathtt{c} \mid \mathtt{s};\ \mathtt{var}\,\mathtt{v_1, v_r := v, v} \\ \models \\ \left(\begin{array}{c}(\mathtt{spar}\ \mathtt{v} \mid \mathtt{v_1} \mid \mathtt{v_r} \mid \mathtt{x1 := x1_1} \bullet \mathtt{A_1[v_1/v]}) \\ \llbracket \mathtt{cs} \rrbracket \\ (\mathtt{spar}\ \mathtt{v} \mid \mathtt{v_r} \mid \mathtt{v_1} \mid \mathtt{x2 := x2_r} \bullet \mathtt{A_2[v_r/v]})\end{array}\right)\end{array}\right)\end{array}} \quad \begin{array}{l} v' = out\alpha s \\ v = x1, x2 \\ fresh\ v_l, v_r \end{array} \qquad (12)$$

As opposed to the transitions for parallelism in the operational semantics, the transitions here lead to change of state before the termination of the parallelism. As defined above, the state is first changed by a declaration of fresh copies $\mathtt{v_1}$ and $\mathtt{v_r}$ of the global variables $\mathtt{v}$. The parallel action $\mathtt{A_1}$ is transformed to record that the original global variables are $\mathtt{v}$, that it uses $\mathtt{v_1}$, but does not use $\mathtt{v_r}$. There is also a record that the variables $\mathtt{x1}$ in its name set take the value of the variables $\mathtt{x1_1}$ upon termination of the parallelism. Finally, the variables $\mathtt{v}$ are renamed to $\mathtt{v_1}$ in $\mathtt{A_1}$. The other action $\mathtt{A_2}$ is transformed in a similar way.

The renaming $\mathtt{A[y/x]}$ substitutes $\mathtt{y}$ for $\mathtt{x}$ in the action $\mathtt{A}$ covering also decorated input, output, and dashed variables, to cater for the uses of $\mathtt{x}$ in schemas and specification statements. For instance, $(\mathtt{x} : [\mathtt{x} > \mathtt{0}, \mathtt{x}' = \mathtt{x} - \mathtt{1}])[\mathtt{y/x}]$ is $\mathtt{y} : [\mathtt{y} > \mathtt{0}, \mathtt{y}' = \mathtt{y} - \mathtt{1}]$ and $[\,\Delta\mathtt{S};\ \mathtt{x!} : \mathbb{Z} \bullet \mathtt{x!} = \mathtt{3}][\mathtt{y/x}]$ is $[\,\Delta\mathtt{S};\ \mathtt{y!} : \mathbb{Z} \bullet \mathtt{y!} = \mathtt{3}]$.

Example 11. We consider again the action *PA* of Example 2. We show the $\Longrightarrow_L$ transitions, and as before in Example 7 repeat the transitions of the operational semantics when no $\Longrightarrow_L$ transition is possible. We define that the first parallel action has control over x, even though neither action actually updates x.

$$\begin{array}{l}(\mathtt{w_0} \in \mathbb{Z} \mid \mathtt{x := w_0} \models \mathtt{PA}) \\ \overset{\mathtt{x:=2}}{\Longrightarrow} \hfill [\text{Rules (4) and (11)}] \\ \left(\begin{array}{l}\mathtt{w_0} \in \mathbb{Z} \wedge \mathtt{w_1} = \mathtt{2} \mid \mathtt{x := w_1} \\ \models \\ \mathtt{Skip};\ \left(\begin{array}{l}(\mathtt{inpA?y} \to \mathtt{int!y} \to \mathtt{out!(y - x)} \to \mathtt{Skip}) \\ \quad \llbracket \{\mathtt{x}\} \mid \{\!\mid \mathtt{int} \mid\!\} \mid \{\} \rrbracket \\ (\mathtt{inpB?z_1} \to \mathtt{int?z_2} \to \mathtt{z_1} > \mathtt{z_2}\ \&\ \mathtt{out!(z_1 - x)} \to \mathtt{Skip})\end{array}\right)\end{array}\right)\end{array}$$

$$\longrightarrow$$

$$\left(\begin{array}{l} \mathtt{w_0} \in \mathbb{Z} \land \mathtt{w_1} = 2 \mid \mathtt{x} := \mathtt{w_1} \\ \models \\ \left(\begin{array}{l} (\mathtt{inpA?y} \to \mathtt{int!y} \to \mathtt{out!}(\mathtt{y} - \mathtt{x}) \to \mathtt{Skip}) \\ \quad [\![\{\mathtt{x}\} \mid \{\!| \, \mathtt{int} \, |\!\} \mid \{\}]\!] \\ (\mathtt{inpB?z_1} \to \mathtt{int?z_2} \to \mathtt{z_1} > \mathtt{z_2} \,\&\, \mathtt{out!}(\mathtt{z_1} - \mathtt{x}) \to \mathtt{Skip}) \end{array}\right) \end{array}\right)$$

$$\overset{\mathtt{var}\, \mathtt{x_1}, \mathtt{x_r} := \mathtt{x}, \mathtt{x}}{\Longrightarrow} \qquad \text{[Rule 12]}$$

$$\left(\begin{array}{l} \mathtt{w_0} \in \mathbb{Z} \land \mathtt{w_1} = 2 \mid \mathtt{x} := \mathtt{w_1};\ \mathtt{var}\, \mathtt{x_1}, \mathtt{x_r} := \mathtt{x}, \mathtt{x} \\ \models \\ \left(\begin{array}{l} (\mathtt{spar}\ \mathtt{x} \mid \mathtt{x_1} \mid \mathtt{x_r} \mid \mathtt{x} := \mathtt{x_1} \bullet \mathtt{inpA?y} \to \mathtt{int!y} \to \mathtt{out!}(\mathtt{y} - \mathtt{x_1}) \to \mathtt{Skip}) \\ \qquad\qquad [\![\{\!| \, \mathtt{int} \, |\!\}]\!] \\ \left(\begin{array}{l} \mathtt{spar}\ \mathtt{x} \mid \mathtt{x_r} \mid \mathtt{x_1} \mid \mathtt{Skip} \bullet \\ \quad \mathtt{inpB?z_1} \to \mathtt{int?z_2} \to \mathtt{z_1} > \mathtt{z_2} \,\&\, \mathtt{out!}(\mathtt{z_1} - \mathtt{x_r}) \to \mathtt{Skip} \end{array}\right) \end{array}\right) \end{array}\right)$$

The second parallel action has write control over no variables, so we write the assignment to the empty list of variables as `Skip`. □

In this work, from the semantics of the new spar construct, we only need the transition rules that allow silent independent evolutions of the parallel actions. The rule that considers evolution of the first parallel action is presented below.

$$\frac{(\mathtt{c} \mid \mathtt{s};\ \mathtt{end}\, \mathtt{v}, \mathtt{y} \models \mathtt{A_1}) \xrightarrow{\epsilon} (\mathtt{c_3} \mid \mathtt{s_3} \models \mathtt{A_3})}{\left(\begin{array}{l} \mathtt{c} \mid \mathtt{s} \\ \models \\ \left(\begin{array}{l} (\mathtt{spar}\ \mathtt{v} \mid \mathtt{x} \mid \mathtt{y} \mid \mathtt{x_1} := \mathtt{z_1} \bullet \mathtt{A_1}) \\ \qquad [\![\mathtt{cs}]\!] \\ (\mathtt{spar}\ \mathtt{v} \mid \mathtt{y} \mid \mathtt{x} \mid \mathtt{x_2} := \mathtt{z_2} \bullet \mathtt{A_2}) \end{array}\right) \end{array}\right) \xrightarrow{\epsilon} \left(\begin{array}{l} \mathtt{c} \mid \mathtt{s_3} \land \mathtt{s};\ \mathtt{end}\, \mathtt{x} \\ \models \\ \left(\begin{array}{l} (\mathtt{spar}\ \mathtt{v} \mid \mathtt{x} \mid \mathtt{y} \mid \mathtt{x_1} := \mathtt{z_1} \bullet \mathtt{A_3}) \\ \qquad [\![\mathtt{cs}]\!] \\ (\mathtt{spar}\ \mathtt{v} \mid \mathtt{y} \mid \mathtt{x} \mid \mathtt{x_2} := \mathtt{z_2} \bullet \mathtt{A_2}) \end{array}\right) \end{array}\right)} \tag{13}$$

The action $\mathtt{A_1}$ is evaluated in the state $\mathtt{s}$ after the original global variables $\mathtt{v}$ and the local variables $\mathtt{y}$ of $\mathtt{A_2}$ are undeclared. The updated state of the parallelism is characterised by the conjunction of the state $\mathtt{s_3}$ reached by $\mathtt{A_1}$, with the original state $\mathtt{s}$, after the local variables $\mathtt{x}$ of $\mathtt{A_1}$ are eliminated. It is important to observe that the input variables of $\mathtt{s_3}$ and $\mathtt{s};\ \mathtt{end}\, \mathtt{x}$ are the same, but their sets of output variables are disjoint, so that conjunction captures the effect of the parallelism. This is akin to the construct for parallelism of designs considered in [18].

Going back to the specification-oriented transition system, independent evolution of the left-hand parallel action $\mathtt{A_1}$ is covered by the following rule. A similar rule caters for evolution of $\mathtt{A_2}$. Like in the operational semantics, the state for $\mathtt{A_1}$ is the global state $\mathtt{s}$, with the global variables $\mathtt{v}$ and the local variables $\mathtt{y}$ of $\mathtt{A_2}$ undeclared. To compose the new state we conjoin the after state $\mathtt{s_3}$ of $\mathtt{A_1}$, with the original state $\mathtt{s}$ followed by the undeclaration of $\mathtt{y}$.

Variables declared in the scope of $\mathtt{A}_1$, as flagged by the label $\mathtt{l}$, are made global, and so they need to be mentioned in the set of variables under the control of $\mathtt{A}_1$ in both parallel actions. Similarly, if the scope of a variable is closed, then it needs to be removed from the set of variables under the control of $\mathtt{A}_1$.

$$\frac{(\mathtt{c} \mid \mathtt{s};\ \mathtt{end}\,\mathtt{v},\mathtt{y} \models \mathtt{A}_1) \stackrel{\mathtt{l}}{\Longrightarrow}_L (\mathtt{c}_3 \mid \mathtt{s}_3 \models \mathtt{A}_3) \quad \mathrm{chan}\,\mathtt{l} = \epsilon \vee \mathrm{chan}\,\mathtt{l} \notin cs}{\begin{array}{l}(\mathtt{c} \mid \mathtt{s} \models (\mathtt{spar}\ \mathtt{v} \mid \mathtt{x} \mid \mathtt{y} \mid \mathtt{x}_1 := \mathtt{z}_1 \bullet \mathtt{A}_1) \llbracket\ \mathtt{cs}\ \rrbracket (\mathtt{spar}\ \mathtt{v} \mid \mathtt{y} \mid \mathtt{x} \mid \mathtt{x}_2 := \mathtt{z}_2 \bullet \mathtt{A}_2)) \\ \stackrel{\mathtt{l}}{\Longrightarrow}_L \\ \left(\begin{array}{l}\mathtt{c}_3 \mid \mathtt{s}_3 \wedge \mathtt{s};\ \mathtt{end}\,\mathtt{x} \\ \models \\ \left(\begin{array}{c}(\mathtt{spar}\ \mathtt{v} \mid \mathtt{x} \upharpoonright (end\,\mathtt{l}), (var\,\mathtt{l}) \mid \mathtt{y} \mid \mathtt{x}_1 := \mathtt{z}_1 \bullet \mathtt{A}_3) \\ \llbracket \mathtt{cs} \rrbracket \\ (\mathtt{spar}\ \mathtt{v} \mid \mathtt{y} \mid \mathtt{x} \upharpoonright (end\,\mathtt{l}), (var\,\mathtt{l}) \mid \mathtt{x}_2 := \mathtt{z}_2 \bullet \mathtt{A}_2)\end{array}\right)\end{array}\right)\end{array}} \tag{14}$$

The function $end\,\mathtt{l}$ gives the variables whose scope are closed in the label $\mathtt{l}$. For example, $end(\mathtt{end}\,\mathtt{x}) = \mathtt{x}$. The function $var\,\mathtt{l}$, on the other hand, gives the variables declared in $\mathtt{l}$. For example, $var(\mathtt{var}\,\mathtt{x}) = \mathtt{x}$ and $var(\mathtt{d?x}) = \mathtt{x}$. Both *end* and *var* are syntactic functions that can be defined by induction on the structure of the actions used in labels in the obvious way. The syntactic function $\mathtt{x} \upharpoonright \mathtt{y}$ removes from the list of variables $\mathtt{x}$ the variables in the list $\mathtt{y}$.

Example 12. Proceeding with the previous example, we have the following sequence of transitions if the left-hand action evolves first.

$\stackrel{\mathtt{inpA?y}}{\Longrightarrow}$ [Rules (7) and (14)]

$$\left(\begin{array}{l}\mathtt{w}_0 \in \mathbb{Z} \wedge \mathtt{w}_1 = 2 \wedge \mathtt{w}_2 \in \mathbb{Z} \mid \mathtt{x} := \mathtt{w}_1;\ \mathtt{var}\,\mathtt{x}_1, \mathtt{x}_r := \mathtt{x}, \mathtt{x};\ \mathtt{var}\,\mathtt{y} := \mathtt{w}_2 \\ \models \\ \left(\begin{array}{l}(\mathtt{spar}\ \mathtt{x} \mid \mathtt{x}_1, \mathtt{y} \mid \mathtt{x}_r \mid \mathtt{x} := \mathtt{x}_1 \bullet (\mathtt{let}\ \mathtt{y} \bullet \mathtt{int!y} \to \mathtt{out!}(\mathtt{y} - \mathtt{x}_1) \to \mathtt{Skip})) \\ \llbracket \{\!|\ \mathtt{int}\ |\!\} \rrbracket \\ \left(\begin{array}{l}\mathtt{spar}\ \mathtt{x} \mid \mathtt{x}_r \mid \mathtt{x}_1, \mathtt{y} \mid \mathtt{Skip} \bullet \\ \quad \mathtt{inpB?z}_1 \to \mathtt{int?z}_2 \to \mathtt{z}_1 > \mathtt{z}_2\ \&\ \mathtt{out!}(\mathtt{z}_1 - \mathtt{x}_r) \to \mathtt{Skip}\end{array}\right)\end{array}\right)\end{array}\right)$$

Above and in what follows, for conciseness, instead of the text actually generated by the application of the transition rules to describe the new state, we give a semantically equivalent, but simpler, description.

$\stackrel{\mathtt{inpB?z}_1}{\Longrightarrow}$ [Rule (7) and Rule similar to (14)]

$$\left(\begin{array}{l}\mathtt{w}_0 \in \mathbb{Z} \wedge \mathtt{w}_1 = 2 \wedge \mathtt{w}_2 \in \mathbb{Z} \wedge \mathtt{w}_3 \in \mathbb{Z} \\ \mid \mathtt{x} := \mathtt{w}_1;\ \mathtt{var}\,\mathtt{x}_1, \mathtt{x}_r := \mathtt{x}, \mathtt{x};\ \mathtt{var}\,\mathtt{y} := \mathtt{w}_2;\ \mathtt{var}\,\mathtt{z}_1 := \mathtt{w}_3 \\ \models \\ \left(\begin{array}{l}(\mathtt{spar}\ \mathtt{x} \mid \mathtt{x}_1, \mathtt{y} \mid \mathtt{x}_r, \mathtt{z}_1 \mid \mathtt{x} := \mathtt{x}_1 \bullet (\mathtt{let}\ \mathtt{y} \bullet \mathtt{int!y} \to \mathtt{out!}(\mathtt{y} - \mathtt{x}_1) \to \mathtt{Skip})) \\ \llbracket \{\!|\ \mathtt{int}\ |\!\} \rrbracket \\ \left(\begin{array}{l}\mathtt{spar}\ \mathtt{x} \mid \mathtt{x}_r, \mathtt{z}_1 \mid \mathtt{x}_1, \mathtt{y} \mid \mathtt{Skip} \bullet \\ \quad (\mathtt{let}\ \mathtt{z}_1 \bullet \mathtt{int?z}_2 \to \mathtt{z}_1 > \mathtt{z}_2\ \&\ \mathtt{out!}(\mathtt{z}_1 - \mathtt{x}_r) \to \mathtt{Skip})\end{array}\right)\end{array}\right)\end{array}\right)$$

□

The rule for synchronisation of an input $\mathtt{d?a}$ with an output $\mathtt{d!e}$ is as follows.

$$\frac{\begin{array}{c}(\mathtt{c} \mid \mathtt{s};\ \mathtt{end}\,\mathtt{v}, \mathtt{y} \models \mathtt{A}_1) \overset{(\mathtt{g}_1, \mathtt{d?a}, \mathtt{LA}_1)}{\Longrightarrow_L} (\mathtt{c}_3 \mid \mathtt{s}_3 \models \mathtt{A}_3) \\ (\mathtt{c} \mid \mathtt{s};\ \mathtt{end}\,\mathtt{v}, \mathtt{x} \models \mathtt{A}_2) \overset{(\mathtt{g}_2, \mathtt{d!e}, \mathtt{LA}_2)}{\Longrightarrow_L} (\mathtt{c}_4 \mid \mathtt{s}_4 \models \mathtt{A}_4) \\ \mathtt{d} \in cs \quad c_3 \wedge c_4 \wedge \exists\, w_0 \bullet (s_3;\ (w_0 = x)) \Leftrightarrow (s_4;\ (w_0 = e))\end{array}}{\begin{array}{l}(\mathtt{c} \mid \mathtt{s} \models (\mathtt{spar}\ \mathtt{v} \mid \mathtt{x} \mid \mathtt{y} \mid \mathtt{x}_1 := \mathtt{z}_1 \bullet \mathtt{A}_1) \,[\![\, \mathtt{cs} \,]\!]\, (\mathtt{spar}\ \mathtt{v} \mid \mathtt{y} \mid \mathtt{x} \mid \mathtt{x}_2 := \mathtt{z}_2 \bullet \mathtt{A}_2)) \\ \overset{(\mathtt{g}_1 \wedge \mathtt{g}_2, \mathtt{d!e}, \mathtt{var\ a} := \mathtt{e};\ \mathtt{LA}_1;\ \mathtt{LA}_2)}{\Longrightarrow_L} \\ \left(\begin{array}{l}\mathtt{c}_3 \wedge \mathtt{c}_4 \wedge \exists\, \mathtt{w}_0 \bullet (\mathtt{s}_3;\ (\mathtt{w}_0 = \mathtt{x})) \Leftrightarrow (\mathtt{s}_4;\ (\mathtt{w}_0 = \mathtt{e})) \mid \mathtt{s}_3 \wedge \mathtt{s}_4 \wedge \mathtt{s};\ \mathtt{end}\,\mathtt{x}, \mathtt{y} \\ \models \\ \left(\begin{array}{c}(\mathtt{spar}\ \mathtt{v} \mid \mathtt{x} \upharpoonright (end\,\mathtt{LA}_1), \mathtt{a}, (var\,\mathtt{LA}_1) \mid \mathtt{y} \upharpoonright (end\,\mathtt{LA}_2), (var\,\mathtt{LA}_2) \mid \mathtt{x}_1 := \mathtt{z}_1 \bullet \mathtt{A}_3) \\ [\![\mathtt{cs}]\!] \\ (\mathtt{spar}\ \mathtt{v} \mid \mathtt{y} \upharpoonright (end\,\mathtt{LA}_2), (var\,\mathtt{LA}_2) \mid \mathtt{x} \upharpoonright (end\,\mathtt{LA}_1), \mathtt{a}, (var\,\mathtt{LA}_1) \mid \mathtt{x}_2 := \mathtt{z}_1 \bullet \mathtt{A}_4)\end{array}\right)\end{array}\right)\end{array}} \quad (15)$$

For the parallelism to progress, both guards in the labels have to be satisfied jointly. As a result of the parallelism, we actually have an output $\mathtt{d!e}$: this is what is observed by the environment of the parallelism. In addition, the input variable $\mathtt{a}$ is declared, and its value is initialized to that of $\mathtt{e}$.

The constraint $\exists\, w_0 \bullet s_3;\ (w_0 = x) \Leftrightarrow s_4;\ (w_0 = e)$ requires that there is a value w_0 that is both the value of $\mathtt{a}$ in the after state s_3 of $\mathtt{A}_1$, and the value of $\mathtt{e}$ in the after state s_4 of $\mathtt{A}_2$. In fact, the value of the expression could be taken in the original state s but an output does not change the state.

The new state is the conjunction of the after states $\mathtt{s}_3$ and $\mathtt{s}_4$ of the parallel actions, and the original state $\mathtt{s}$ where the local versions $\mathtt{x}$ and $\mathtt{y}$ of the original global variables are all undeclared. This is necessary because neither $\mathtt{s}_3$ nor $\mathtt{s}_4$ includes the original global variables. On the other hand, in $\mathtt{s};\ \mathtt{end}\,\mathtt{x}, \mathtt{y}$, these are the only output variables in scope. So, the conjunction is between predicates with the same input variables, but disjoint sets of output variables.

As for the previous transition rule, variables declared or undeclared, as stated in the labels, are recorded in the appropriate sets of variables of the parallel actions. These include the implicit declaration of the input variable $\mathtt{a}$, and the variables declared or undeclared in the actions $\mathtt{LA}_1$ and $\mathtt{LA}_2$ of the labels.

We omit the similar rules for synchronisation of an output and an input, two inputs, or two outputs. For two inputs $\mathtt{d?a}$ and $\mathtt{d?b}$, one of the input variables $\mathtt{a}$ is implicitly declared by the input event, and the other $\mathtt{b}$ is declared explicitly, and initialised to $\mathtt{a}$. In the case of two outputs $\mathtt{d!e}$ and $\mathtt{d!f}$, there are no variable declarations. The output value is that of $\mathtt{e}$, and the guard guarantees that $\mathtt{e} = \mathtt{f}$.

Example 13. Proceeding with our example, we have the synchronisation.

$\stackrel{(\mathtt{int!y},\mathtt{var}\,\mathtt{z_2}:=\mathtt{y})}{\Longrightarrow}$ [Rules (6), (10), (7), and similar to (15)]

$$\left(\begin{array}{l} \mathtt{w_0} \in \mathbb{Z} \wedge \mathtt{w_1} = 2 \wedge \mathtt{w_2} \in \mathbb{Z} \wedge \mathtt{w_3} \in \mathbb{Z} \wedge \mathtt{w_4} = \mathtt{w_2} \wedge \mathtt{w_5} \in \mathbb{Z} \wedge \mathtt{w_2} = \mathtt{w_5} \\ \mid \mathtt{x} := \mathtt{w_1};\ \mathtt{var}\,\mathtt{x_1}, \mathtt{x_r} := \mathtt{x}, \mathtt{x};\ \mathtt{var}\,\mathtt{y}, \mathtt{z_1}, \mathtt{z_2} := \mathtt{w_2}, \mathtt{w_3}, \mathtt{w_5} \\ \models \\ \left(\begin{array}{l} (\mathtt{spar}\ \mathtt{x} \mid \mathtt{x_1}, \mathtt{y} \mid \mathtt{x_r}, \mathtt{z_1}, \mathtt{z_2} \mid \mathtt{x} := \mathtt{x_1} \bullet (\mathtt{let}\ \mathtt{y} \bullet \mathtt{out!}(\mathtt{y} - \mathtt{x_1}) \to \mathtt{Skip})) \\ \qquad\qquad [\![\{\!\mid \mathtt{int} \mid\!\}]\!] \\ \left(\begin{array}{l} \mathtt{spar}\ \mathtt{x} \mid \mathtt{x_r}, \mathtt{z_1}, \mathtt{z_2} \mid \mathtt{x_1}, \mathtt{y} \mid \mathtt{Skip} \bullet \\ \quad (\mathtt{let}\ \mathtt{z_1}, \mathtt{z_2} \bullet \mathtt{z_1} > \mathtt{z_2}\ \&\ \mathtt{out!}(\mathtt{z_1} - \mathtt{x_r}) \to \mathtt{Skip}) \end{array}\right) \end{array}\right) \end{array}\right)$$

Again, the parallel actions can both evolve independently. We consider below one order of evolution: the first action evolves first.

$\stackrel{\mathtt{out!}(\mathtt{y}-\mathtt{x_1})}{\Longrightarrow}$ [Rules (6), (10), and (14)]

$$\left(\begin{array}{l} \left(\begin{array}{l} \mathtt{w_0} \in \mathbb{Z} \wedge \mathtt{w_1} = 2 \wedge \mathtt{w_2} \in \mathbb{Z} \wedge \mathtt{w_3} \in \mathbb{Z} \wedge \\ \mathtt{w_4} = \mathtt{w_2} \wedge \mathtt{w_5} \in \mathbb{Z} \wedge \mathtt{w_2} = \mathtt{w_5} \wedge \mathtt{w_6} = \mathtt{w_2} - \mathtt{w_1} \end{array}\right) \\ \mid \mathtt{x} := \mathtt{w_1};\ \mathtt{var}\,\mathtt{x_1}, \mathtt{x_r} := \mathtt{x}, \mathtt{x};\ \mathtt{var}\,\mathtt{y}, \mathtt{z_1}, \mathtt{z_2} := \mathtt{w_2}, \mathtt{w_3}, \mathtt{w_5} \\ \models \\ \left(\begin{array}{l} (\mathtt{spar}\ \mathtt{x} \mid \mathtt{x_1}, \mathtt{y} \mid \mathtt{x_r}, \mathtt{z_1}, \mathtt{z_2} \mid \mathtt{x} := \mathtt{x_1} \bullet (\mathtt{let}\ \mathtt{y} \bullet \mathtt{Skip})) \\ \qquad\qquad [\![\{\!\mid \mathtt{int} \mid\!\}]\!] \\ \left(\begin{array}{l} \mathtt{spar}\ \mathtt{x} \mid \mathtt{x_r}, \mathtt{z_1}, \mathtt{z_2} \mid \mathtt{x_1}, \mathtt{y} \mid \mathtt{Skip} \bullet \\ \quad (\mathtt{let}\ \mathtt{z_1}, \mathtt{z_2} \bullet \mathtt{z_1} > \mathtt{z_2}\ \&\ \mathtt{out!}(\mathtt{z_1} - \mathtt{x_r}) \to \mathtt{Skip}) \end{array}\right) \end{array}\right) \end{array}\right)$$

$\stackrel{\mathtt{z_1}>\mathtt{z_2}}{\Longrightarrow}$ [Rules (5), (10), and similar to (14)]

$$\left(\begin{array}{l} \left(\begin{array}{l} \mathtt{w_0} \in \mathbb{Z} \wedge \mathtt{w_1} = 2 \wedge \mathtt{w_2} \in \mathbb{Z} \wedge \mathtt{w_3} \in \mathbb{Z} \wedge \\ \mathtt{w_4} = \mathtt{w_2} \wedge \mathtt{w_5} \in \mathbb{Z} \wedge \mathtt{w_2} = \mathtt{w_5} \wedge \mathtt{w_6} = \mathtt{w_2} - \mathtt{w_1} \wedge \mathtt{w_3} > \mathtt{w_5} \end{array}\right) \\ \mid \mathtt{x} := \mathtt{w_1};\ \mathtt{var}\,\mathtt{x_1}, \mathtt{x_r} := \mathtt{x}, \mathtt{x};\ \mathtt{var}\,\mathtt{y}, \mathtt{z_1}, \mathtt{z_2} := \mathtt{w_2}, \mathtt{w_3}, \mathtt{w_5} \\ \models \\ \left(\begin{array}{l} (\mathtt{spar}\ \mathtt{x} \mid \mathtt{x_1}, \mathtt{y} \mid \mathtt{x_r}, \mathtt{z_1}, \mathtt{z_2} \mid \mathtt{x} := \mathtt{x_1} \bullet (\mathtt{let}\ \mathtt{y} \bullet \mathtt{Skip})) \\ \qquad\qquad [\![\{\!\mid \mathtt{int} \mid\!\}]\!] \\ \left(\begin{array}{l} \mathtt{spar}\ \mathtt{x} \mid \mathtt{x_r}, \mathtt{z_1}, \mathtt{z_2} \mid \mathtt{x_1}, \mathtt{y} \mid \mathtt{Skip} \bullet \\ \quad (\mathtt{let}\ \mathtt{z_1}, \mathtt{z_2} \bullet \mathtt{out!}(\mathtt{z_1} - \mathtt{x_r}) \to \mathtt{Skip}) \end{array}\right) \end{array}\right) \end{array}\right)$$

$\stackrel{\mathtt{out!}(\mathtt{z_1}-\mathtt{x_r})}{\Longrightarrow}$ [Rules (6), (10), and similar to (14)]

$$\left(\begin{array}{l} \left(\begin{array}{l} \mathtt{w_0} \in \mathbb{Z} \wedge \mathtt{w_1} = 2 \wedge \mathtt{w_2} \in \mathbb{Z} \wedge \mathtt{w_3} \in \mathbb{Z} \wedge \\ \mathtt{w_4} = \mathtt{w_2} \wedge \mathtt{w_5} \in \mathbb{Z} \wedge \mathtt{w_2} = \mathtt{w_5} \wedge \mathtt{w_6} = \mathtt{w_2} - \mathtt{w_1} \wedge \mathtt{w_3} > \mathtt{w_5} \wedge \mathtt{w_7} = \mathtt{w_3} - \mathtt{w_1} \end{array}\right) \\ \mid \mathtt{x} := \mathtt{w_1};\ \mathtt{var}\,\mathtt{x_1}, \mathtt{x_r} := \mathtt{x}, \mathtt{x};\ \mathtt{var}\,\mathtt{y}, \mathtt{z_1}, \mathtt{z_2} := \mathtt{w_2}, \mathtt{w_3}, \mathtt{w_5} \\ \models \\ \left(\begin{array}{l} (\mathtt{spar}\ \mathtt{x} \mid \mathtt{x_1}, \mathtt{y} \mid \mathtt{x_r}, \mathtt{z_1}, \mathtt{z_2} \mid \mathtt{x} := \mathtt{x_1} \bullet (\mathtt{let}\ \mathtt{y} \bullet \mathtt{Skip})) \\ \qquad\qquad [\![\{\!\mid \mathtt{int} \mid\!\}]\!] \\ (\mathtt{spar}\ \mathtt{x} \mid \mathtt{x_r}, \mathtt{z_1}, \mathtt{z_2} \mid \mathtt{x_1}, \mathtt{y} \mid \mathtt{Skip} \bullet (\mathtt{let}\ \mathtt{z_1}, \mathtt{z_2} \bullet \mathtt{Skip})) \end{array}\right) \end{array}\right)$$

$\overset{\mathtt{end\,y}}{\Longrightarrow}$ [Rules (10) and (14)]

$$\left(\begin{array}{l}\left(\begin{array}{l}\mathtt{w_0} \in \mathbb{Z} \wedge \mathtt{w_1} = 2 \wedge \mathtt{w_2} \in \mathbb{Z} \wedge \mathtt{w_3} \in \mathbb{Z} \wedge \\ \mathtt{w_4} = \mathtt{w_2} \wedge \mathtt{w_5} \in \mathbb{Z} \wedge \mathtt{w_2} = \mathtt{w_5} \wedge \mathtt{w_6} = \mathtt{w_2} - \mathtt{w_1} \wedge \mathtt{w_3} > \mathtt{w_5} \wedge \mathtt{w_7} = \mathtt{w_3} - \mathtt{w_1}\end{array}\right) \\ \mid \mathtt{x} := \mathtt{w_1};\ \mathtt{var}\,\mathtt{x_1}, \mathtt{x_r} := \mathtt{x}, \mathtt{x};\ \mathtt{var}\,\mathtt{y}, \mathtt{z_1}, \mathtt{z_2} := \mathtt{w_2}, \mathtt{w_3}, \mathtt{w_5};\ \mathtt{end}\,\mathtt{y} \\ \models \\ \left(\begin{array}{c}(\mathtt{spar}\ \mathtt{x} \mid \mathtt{x_1} \mid \mathtt{x_r}, \mathtt{z_1}, \mathtt{z_2} \mid \mathtt{x} := \mathtt{x_1} \bullet \mathtt{Skip}) \\ [\![\{\!| \mathtt{int} |\!\}]\!] \\ (\mathtt{spar}\ \mathtt{x} \mid \mathtt{x_r}, \mathtt{z_1}, \mathtt{z_2} \mid \mathtt{x_1} \mid \mathtt{Skip} \bullet (\mathtt{let}\ \mathtt{z_1}, \mathtt{z_2} \bullet \mathtt{Skip}))\end{array}\right)\end{array}\right)$$

$\overset{\mathtt{end}\,\mathtt{z_1},\mathtt{z_2}}{\Longrightarrow}$ [Rules (10) and similar to (14)]

$$\left(\begin{array}{l}\left(\begin{array}{l}\mathtt{w_0} \in \mathbb{Z} \wedge \mathtt{w_1} = 2 \wedge \mathtt{w_2} \in \mathbb{Z} \wedge \mathtt{w_3} \in \mathbb{Z} \wedge \\ \mathtt{w_4} = \mathtt{w_2} \wedge \mathtt{w_5} \in \mathbb{Z} \wedge \mathtt{w_2} = \mathtt{w_5} \wedge \mathtt{w_6} = \mathtt{w_2} - \mathtt{w_1} \wedge \mathtt{w_3} > \mathtt{w_5} \wedge \mathtt{w_7} = \mathtt{w_3} - \mathtt{w_1}\end{array}\right) \\ \mid \left(\begin{array}{l}\mathtt{x} := \mathtt{w_1};\ \mathtt{var}\,\mathtt{x_1}, \mathtt{x_r} := \mathtt{x}, \mathtt{x}; \\ \mathtt{var}\,\mathtt{y}, \mathtt{z_1}, \mathtt{z_2} := \mathtt{w_2}, \mathtt{w_3}, \mathtt{w_5};\ \mathtt{end}\,\mathtt{y};\ \mathtt{end}\,\mathtt{z_1}, \mathtt{z_2}\end{array}\right) \\ \models \\ (\mathtt{spar}\ \mathtt{x} \mid \mathtt{x_1} \mid \mathtt{x_r} \mid \mathtt{x} := \mathtt{x_1} \bullet \mathtt{Skip})\ [\![\{\!| \mathtt{int} |\!\}]\!]\ (\mathtt{spar}\ \mathtt{x} \mid \mathtt{x_r} \mid \mathtt{x_1} \mid \mathtt{Skip} \bullet \mathtt{Skip})\end{array}\right)$$

□

The rule that applies when both parallel actions terminate is as follows. The label records the changes to the state.

$$\frac{c}{\left(\begin{array}{l}\mathtt{c} \mid \mathtt{s} \\ \models \\ \left(\begin{array}{c}(\mathtt{spar}\ \mathtt{v} \mid \mathtt{x}, \mathtt{z_1} \mid \mathtt{y}, \mathtt{z_2} \mid \mathtt{x_1} := \mathtt{z_1} \bullet \mathtt{Skip}) \\ [\![\mathtt{cs}]\!] \\ (\mathtt{spar}\ \mathtt{v} \mid \mathtt{y}, \mathtt{z_2} \mid \mathtt{x}, \mathtt{z_1} \mid \mathtt{x_2} := \mathtt{z_2} \bullet \mathtt{Skip})\end{array}\right)\end{array}\right)}$$

$$\overset{\mathtt{x_1},\mathtt{x_2}:=\mathtt{z_1},\mathtt{z_2};\ \mathtt{end}\,\mathtt{x},\mathtt{z_1},\mathtt{y},\mathtt{z_2}}{\Longrightarrow_L} \tag{16}$$

$$(\mathtt{c} \mid \mathtt{s};\ \mathtt{x_1}, \mathtt{x_2} := \mathtt{z_1}, \mathtt{z_2};\ \mathtt{end}\,\mathtt{x}, \mathtt{z_1}, \mathtt{y}, \mathtt{z_2} \models \mathtt{Skip})$$

In the final state of the parallelism, the local versions $\mathtt{x}$ and $\mathtt{y}$ of the global variables are undeclared after they are used to update the global variables.

Example 14. We can now conclude our running example.

$\overset{\mathtt{x}:=\mathtt{x_1};\ \mathtt{end}\,\mathtt{x_1},\mathtt{x_r}}{\Longrightarrow}$ [Rule (16)]

$$\left(\begin{array}{l}\left(\begin{array}{l}\mathtt{w_0} \in \mathbb{Z} \wedge \mathtt{w_1} = 2 \wedge \mathtt{w_2} \in \mathbb{Z} \wedge \mathtt{w_3} \in \mathbb{Z} \wedge \\ \mathtt{w_4} = \mathtt{w_2} \wedge \mathtt{w_5} \in \mathbb{Z} \wedge \mathtt{w_2} = \mathtt{w_5} \wedge \mathtt{w_6} = \mathtt{w_2} - \mathtt{w_1} \wedge \mathtt{w_3} > \mathtt{w_5} \wedge \mathtt{w_7} = \mathtt{w_3} - \mathtt{w_1}\end{array}\right) \\ \mid \left(\begin{array}{l}\mathtt{x} := \mathtt{w_1};\ \mathtt{var}\,\mathtt{x_1}, \mathtt{x_r} := \mathtt{x}, \mathtt{x}; \\ \mathtt{var}\,\mathtt{y}, \mathtt{z_1}, \mathtt{z_2} := \mathtt{w_2}, \mathtt{w_3}, \mathtt{w_5};\ \mathtt{end}\,\mathtt{y};\ \mathtt{end}\mathtt{z_1}, \mathtt{z_2};\ \mathtt{x} := \mathtt{x_1};\ \mathtt{end}\,\mathtt{x_1}, \mathtt{x_r}\end{array}\right) \\ \models \\ \mathtt{Skip}\end{array}\right)$$

It is not difficult to prove that the final state of the parallelism is equivalent to

$x := w_1$; its alphabet includes only x (and x'). □

The rules for external choice are not given here.

Hiding There are two rules for hiding. The first allows evolution of the action to which the hiding is applied to lead to evolution of the hiding. This occurs when the evolution is not via a communication through a hidden channel.

$$\frac{(\mathtt{c}_1 \mid \mathtt{s}_1 \models \mathtt{A}_1) \stackrel{\mathtt{l}}{\Longrightarrow}_L (\mathtt{c}_2 \mid \mathtt{s}_2 \models \mathtt{A}_2) \quad \text{chan } \mathtt{l} \notin cs}{(\mathtt{c}_1 \mid \mathtt{s}_1 \models \mathtt{A}_1 \setminus \mathtt{cs}) \stackrel{\mathtt{l}}{\Longrightarrow}_L (\mathtt{c}_2 \mid \mathtt{s}_2 \models \mathtt{A}_2 \setminus \mathtt{cs})} \tag{17}$$

The second rule is for when the communication is through a hidden channel. In this case, the communication disappears. The evolution, therefore, is only possible if the guard is not `True` and the action is not `Skip`. In this case, we do not have the possibility of introducing a silent transition.

$$\frac{\begin{array}{c}(\mathtt{c}_1 \mid \mathtt{s}_1 \models \mathtt{A}_1) \stackrel{(\mathtt{g},\mathtt{e},\mathtt{A})}{\Longrightarrow}_L (\mathtt{c}_2 \mid \mathtt{s}_2 \models \mathtt{A}_2) \\ (\mathtt{g} \neq \mathtt{True} \vee \mathtt{A} \neq \mathtt{Skip}) \wedge (\text{chan } \mathtt{e} = \epsilon \vee \text{chan } \mathtt{e} \in cs)\end{array}}{(\mathtt{c}_1 \mid \mathtt{s}_1 \models \mathtt{A}_1 \setminus \mathtt{cs}) \stackrel{(\mathtt{g},\epsilon,\mathtt{A})}{\Longrightarrow}_L (\mathtt{c}_2 \mid \mathtt{s}_2 \models \mathtt{A}_2 \setminus \mathtt{cs})} \tag{18}$$

Transitions of the operational semantics that are truly silent in the sense of the $\Longrightarrow_L$ relation, so that they do not entail any guards, communications, or state changes, are considered in the next section.

3.3 Silent transitions

As already explained, the specification-oriented transition system has no silent transitions. In the previous section, we have defined a transition relation $\Longrightarrow_L$ which indeed has no silent transitions, but is not defined for some configurations for which there is a transition in the operational semantics. For instance, in Example 7, the second transition and a few others are transitions of the operational semantics. There are no corresponding transitions for $\Longrightarrow_L$.

We proceed with the definition of the specification-oriented transition system by introducing a new transition relation $(\mathtt{c}_1 \mid \mathtt{s}_1 \models \mathtt{A}_1) \stackrel{(\mathtt{g},\mathtt{e},\mathtt{A})}{\Longrightarrow}_{SR} (\mathtt{c}_2 \mid \mathtt{s}_2 \models \mathtt{A}_2)$. It associates a configuration $(\mathtt{c}_1 \mid \mathtt{s}_1 \models \mathtt{A}_1)$ to a configuration $(\mathtt{c}_2 \mid \mathtt{s}_2 \models \mathtt{A}_2)$ if, by starting from $(\mathtt{c}_1 \mid \mathtt{s}_1 \models \mathtt{A}_1)$, following a transition from $\Longrightarrow_L$, and then as many silent transitions $\stackrel{\epsilon}{\longrightarrow}$ as possible, we reach $(\mathtt{c}_2 \mid \mathtt{s}_2 \models \mathtt{A}_2)$.

By considering as many silent transitions as possible, we ensure that a configuration $(\mathtt{c}_1 \mid \mathtt{s}_1 \models \mathtt{A}_1)$ is related only to those configurations $(\mathtt{c}_2 \mid \mathtt{s}_2 \models \mathtt{A}_2)$ that can be reached after as much internal progress as possible has been made. For testing, extra transitions that represent partial internal progress are of no value. They would give rise to useless tests, and are avoided here.

To define the new relation $(\mathtt{c}_1 \mid \mathtt{s}_1 \models \mathtt{A}_1) \stackrel{(\mathtt{g},\mathtt{e},\mathtt{A})}{\Longrightarrow}_{SR} (\mathtt{c}_2 \mid \mathtt{s}_2 \models \mathtt{A}_2)$, we consider first the transitive closure $\longrightarrow^{\epsilon *}$ of the transition relation $\longrightarrow$ of the operational

semantics when restricted to silent transitions with no corresponding transition in $\Longrightarrow_L$. It is defined by the two transition rules in the sequel.

$$\frac{(\mathtt{c}_1 \mid \mathtt{s}_1 \models \mathtt{A}_1) \xrightarrow{\epsilon} (\mathtt{c}_2 \mid \mathtt{s}_2 \models \mathtt{A}_2) \quad (\mathtt{c}_1 \mid \mathtt{s}_1 \models \mathtt{A}_1) \not\Rightarrow_L (\mathtt{c}_2 \mid \mathtt{s}_2 \models \mathtt{A}_2)}{(\mathtt{c}_1 \mid \mathtt{s}_1 \models \mathtt{A}_1) \longrightarrow^{\epsilon *} (\mathtt{c}_2 \mid \mathtt{s}_2 \models \mathtt{A}_2)} \tag{19}$$

In the above rule, we write $(\mathtt{c}_1 \mid \mathtt{s}_1 \models \mathtt{A}_1) \not\Rightarrow_L (\mathtt{c}_2 \mid \mathtt{s}_2 \models \mathtt{A}_2)$ as an abbreviation for $\neg\ \exists \mathtt{l} \bullet (\mathtt{c}_1 \mid \mathtt{s}_1 \models \mathtt{A}_1) \stackrel{\mathtt{l}}{\Longrightarrow}_L (\mathtt{c}_2 \mid \mathtt{s}_2 \models \mathtt{A}_2)$. We require that there is no specification-oriented transition from $(\mathtt{c}_1 \mid \mathtt{s}_1 \models \mathtt{A}_1)$ to $(\mathtt{c}_2 \mid \mathtt{s}_2 \models \mathtt{A}_2)$ because many of the silent transitions of the operational semantics correspond to (non-silent) transitions of the specification-oriented system. For instance, the transitions for assignment are silent in the operational semantics, but not in the specification-oriented system. What we want is to ignore transitions that genuinely provide no information is terms of guards, events, or action execution. Examples are the transitions for internal choice (see Rules (11) in Appendix A).

The second transition rule allows the composition of silent transitions.

$$\frac{\begin{array}{c}(\mathtt{c}_1 \mid \mathtt{s}_1 \models \mathtt{A}_1) \longrightarrow^{\epsilon *} (\mathtt{c}_2 \mid \mathtt{s}_2 \models \mathtt{A}_2) \\ (\mathtt{c}_2 \mid \mathtt{s}_2 \models \mathtt{A}_2) \xrightarrow{\epsilon} (\mathtt{c}_3 \mid \mathtt{s}_3 \models \mathtt{A}_3) \quad (\mathtt{c}_2 \mid \mathtt{s}_2 \models \mathtt{A}_2) \not\Rightarrow_L (\mathtt{c}_3 \mid \mathtt{s}_3 \models \mathtt{A}_3)\end{array}}{(\mathtt{c}_1 \mid \mathtt{s}_1 \models \mathtt{A}_1) \longrightarrow^{\epsilon *} (\mathtt{c}_3 \mid \mathtt{s}_3 \models \mathtt{A}_3)} \tag{20}$$

We again check that the transitions composed are truly silent.

Example 15. In the context of our example action E, we have the following.

$$\begin{array}{l}\left(\begin{array}{l}\mathtt{w}_0 \in \mathbb{Z} \land \mathtt{w}_1 \in \mathbb{Z} \land \mathtt{w}_2 = 2 \mid \mathtt{x}, \mathtt{y} := \mathtt{w}_2, \mathtt{w}_1 \\ \models \\ \mathtt{Skip};\ (\mathtt{y} > \mathtt{x}\ \&\ \mathtt{out!}(\mathtt{y} - \mathtt{x}) \rightarrow \mathtt{Skip} \sqcap \mathtt{inp?z} \rightarrow \mathtt{Stop});\ \mathtt{x} := \mathtt{y}\end{array}\right) \\ \longrightarrow^{\epsilon *} \\ \left(\begin{array}{l}\mathtt{w}_0 \in \mathbb{Z} \land \mathtt{w}_1 \in \mathbb{Z} \land \mathtt{w}_2 = 2 \mid \mathtt{x}, \mathtt{y} := \mathtt{w}_2, \mathtt{w}_1 \\ \models \\ (\mathtt{y} > \mathtt{x}\ \&\ \mathtt{out!}(\mathtt{y} - \mathtt{x}) \rightarrow \mathtt{Skip});\ \mathtt{x} := \mathtt{y}\end{array}\right)\end{array}$$

This corresponds to choosing the first action of the internal choice. For a choice of the second action, we have the transition below.

$$\begin{array}{l}\left(\begin{array}{l}\mathtt{w}_0 \in \mathbb{Z} \land \mathtt{w}_1 \in \mathbb{Z} \land \mathtt{w}_2 = 2 \mid \mathtt{x}, \mathtt{y} := \mathtt{w}_2, \mathtt{w}_1 \\ \models \\ \mathtt{Skip};\ (\mathtt{y} > \mathtt{x}\ \&\ \mathtt{out!}(\mathtt{y} - \mathtt{x}) \rightarrow \mathtt{Skip} \sqcap \mathtt{inp?z} \rightarrow \mathtt{Stop});\ \mathtt{x} := \mathtt{y}\end{array}\right) \\ \longrightarrow^{\epsilon *} \\ \left(\begin{array}{l}\mathtt{w}_0 \in \mathbb{Z} \land \mathtt{w}_1 \in \mathbb{Z} \land \mathtt{w}_2 = 2 \mid \mathtt{x}, \mathtt{y} := \mathtt{w}_2, \mathtt{w}_1 \\ \models \\ (\mathtt{inp?z} \rightarrow \mathtt{Stop});\ \mathtt{x} := \mathtt{y}\end{array}\right)\end{array}$$

We also have the transition below.

$$\left(\begin{array}{l} \mathtt{w_0} \in \mathbb{Z} \land \mathtt{w_1} \in \mathbb{Z} \land \mathtt{w_2} = 2 \land \mathtt{w_1} > \mathtt{w_2} \land \mathtt{w_3} = \mathtt{w_1} - \mathtt{w_2} \mid \mathtt{x}, \mathtt{y} := \mathtt{w_2}, \mathtt{w_1} \\ \models \\ \mathtt{Skip};\ \mathtt{x} := \mathtt{y} \end{array}\right)$$
$$\longrightarrow^{\epsilon *}$$
$$\left(\begin{array}{l} \mathtt{w_0} \in \mathbb{Z} \land \mathtt{w_1} \in \mathbb{Z} \land \mathtt{w_2} = 2 \land \mathtt{w_1} > \mathtt{w_2} \land \mathtt{w_3} = \mathtt{w_1} - \mathtt{w_2} \mid \mathtt{x}, \mathtt{y} := \mathtt{w_2}, \mathtt{w_1} \\ \models \\ \mathtt{x} := \mathtt{y} \end{array}\right)$$

This corresponds to a single silent transition of the operational semantics. □

The new relation $(\mathtt{c_1} \mid \mathtt{s_1} \models \mathtt{A_1}) \stackrel{(\mathtt{g},\mathtt{e},\mathtt{A})}{\Longrightarrow}_{SR} (\mathtt{c_2} \mid \mathtt{s_2} \models \mathtt{A_2})$ is defined below.

$$\frac{\begin{array}{c}(\mathtt{c_1} \mid \mathtt{s_1} \models \mathtt{A_1}) \stackrel{1}{\Longrightarrow}_L (\mathtt{c_2} \mid \mathtt{s_2} \models \mathtt{A_2}) \\ (\mathtt{c_2} \mid \mathtt{s_2} \models \mathtt{A_2}) \longrightarrow^{\epsilon *} (\mathtt{c_3} \mid \mathtt{s_3} \models \mathtt{A_3}) \qquad (\mathtt{c_3} \mid \mathtt{s_3} \models \mathtt{A_3}) \nrightarrow^{\epsilon *}\end{array}}{(\mathtt{c_1} \mid \mathtt{s_1} \models \mathtt{A_1}) \stackrel{1}{\Longrightarrow}_{SR} (\mathtt{c_3} \mid \mathtt{s_3} \models \mathtt{A_3})} \tag{21}$$

We write $(\mathtt{c_1} \mid \mathtt{s_1} \models \mathtt{A_1}) \nrightarrow^{\epsilon *}$ when $(\mathtt{c_1} \mid \mathtt{s_1} \models \mathtt{A_1})$ is a stuck configuration with respect to $\longrightarrow^{\epsilon *}$, that is, when $\neg\ \exists\, \mathtt{c_2}, \mathtt{s_2}, \mathtt{A_2} \bullet (\mathtt{c_1} \mid \mathtt{s_1} \models \mathtt{A_1}) \longrightarrow^{\epsilon *} (\mathtt{c_2} \mid \mathtt{s_2} \models \mathtt{A_2})$.

Since the configurations of the specification-oriented transition system are the same as those of the *Circus* operational semantics, we can combine their transition relations in a simple way. This has already been indicated in Example 7, where we consider the two transition relations for a single example.

It is possible that a $\Longrightarrow_L$ transition is followed by no $\longrightarrow^{\epsilon *}$ transitions. In this case the $\Longrightarrow_L$ transition corresponds to a $\Longrightarrow_{SR}$ transition.

$$\frac{(\mathtt{c_1} \mid \mathtt{s_1} \models \mathtt{A_1}) \stackrel{1}{\Longrightarrow}_L (\mathtt{c_2} \mid \mathtt{s_2} \models \mathtt{A_2}) \quad (\mathtt{c_2} \mid \mathtt{s_2} \models \mathtt{A_2}) \nrightarrow^{\epsilon *}}{(\mathtt{c_1} \mid \mathtt{s_1} \models \mathtt{A_1}) \stackrel{1}{\Longrightarrow}_{SR} (\mathtt{c_2} \mid \mathtt{s_2} \models \mathtt{A_2})} \tag{22}$$

Example 16. Following from Examples 7 and 15, we can use the rules above to justify the following transitions for our example action E. Again, we present separately the two paths arising from the internal choice.

$$(\mathtt{w_0} \in \mathbb{Z} \land \mathtt{w_1} \in \mathbb{Z} \mid \mathtt{x}, \mathtt{y} := \mathtt{w_0}, \mathtt{w_1} \models \mathtt{E})$$
$$\stackrel{\mathtt{x}:=2}{\Longrightarrow}_{SR}$$
$$\left(\begin{array}{l} \mathtt{w_0} \in \mathbb{Z} \land \mathtt{w_1} \in \mathbb{Z} \land \mathtt{w_2} = 2 \mid \mathtt{x}, \mathtt{y} := \mathtt{w_2}, \mathtt{w_1} \\ \models \\ (\mathtt{y} > \mathtt{x}\ \&\ \mathtt{out!}(\mathtt{y} - \mathtt{x}) \rightarrow \mathtt{Skip});\ \mathtt{x} := \mathtt{y} \end{array}\right)$$
$$\stackrel{\mathtt{y}>\mathtt{x}}{\Longrightarrow}_{SR}$$
$$\left(\begin{array}{l} \mathtt{w_0} \in \mathbb{Z} \land \mathtt{w_1} \in \mathbb{Z} \land \mathtt{w_2} = 2 \land \mathtt{w_1} > \mathtt{w_2} \mid \mathtt{x}, \mathtt{y} := \mathtt{w_2}, \mathtt{w_1} \\ \models \\ (\mathtt{out!}(\mathtt{y} - \mathtt{x}) \rightarrow \mathtt{Skip});\ \mathtt{x} := \mathtt{y} \end{array}\right)$$

$$\stackrel{\mathtt{out!(y-x)}}{\Longrightarrow_{SR}}$$
$$\left(\begin{array}{l} \mathtt{w_0} \in \mathbb{Z} \land \mathtt{w_1} \in \mathbb{Z} \land \mathtt{w_2} = 2 \land \mathtt{w_1} > \mathtt{w_2} \land \mathtt{w_3} = \mathtt{w_1} - \mathtt{w_2} \mid \mathtt{x}, \mathtt{y} := \mathtt{w_2}, \mathtt{w_1} \\ \models \\ \mathtt{x} := \mathtt{y} \end{array}\right)$$
$$\stackrel{\mathtt{x:=y}}{\Longrightarrow_{SR}}$$
$$\left(\begin{array}{l} \mathtt{w_0} \in \mathbb{Z} \land \mathtt{w_1} \in \mathbb{Z} \land \mathtt{w_2} = 2 \land \mathtt{w_1} > \mathtt{w_2} \land \mathtt{w_3} = \mathtt{w_1} - \mathtt{w_2} \land \mathtt{w_4} = \mathtt{w_2} \mid \mathtt{x}, \mathtt{y} := \mathtt{w_4}, \mathtt{w_1} \\ \models \\ \mathtt{Skip} \end{array}\right)$$

For the second option of the internal choice, we proceed as follows.

$$(\mathtt{w_0} \in \mathbb{Z} \land \mathtt{w_1} \in \mathbb{Z} \mid \mathtt{x}, \mathtt{y} := \mathtt{w_0}, \mathtt{w_1} \models \mathtt{E})$$
$$\stackrel{\mathtt{x:=2}}{\Longrightarrow_{SR}}$$
$$\left(\begin{array}{l} \mathtt{w_0} \in \mathbb{Z} \land \mathtt{w_1} \in \mathbb{Z} \land \mathtt{w_2} = 2 \mid \mathtt{x}, \mathtt{y} := \mathtt{w_2}, \mathtt{w_1} \\ \models \\ (\mathtt{inp?z} \to \mathtt{Stop});\ \mathtt{x} := \mathtt{y} \end{array}\right)$$
$$\stackrel{\mathtt{inp?z}}{\Longrightarrow_{SR}}$$
$$\left(\begin{array}{l} \mathtt{w_0} \in \mathbb{Z} \land \mathtt{w_1} \in \mathbb{Z} \land \mathtt{w_2} = 2 \land \mathtt{w_3} \in \mathbb{Z} \mid \mathtt{x}, \mathtt{y} := \mathtt{w_2}, \mathtt{w_1};\ \mathtt{var\,z} := \mathtt{w_3} \\ \models \\ (\mathtt{let\ z} \bullet \mathtt{Stop});\ \mathtt{x} := \mathtt{y} \end{array}\right)$$

From here, we cannot proceed once again. □

If the behaviour of an action as described by the operational semantics starts with (truly) silent transitions, then $\Longrightarrow_{SR}$ cannot give a complete account of its execution, because it does not consider leading silent transitions.

Example 17. We consider the transitions below.

$$(\mathtt{w_0} \in \mathbb{Z} \land \mathtt{w_1} \in \mathbb{Z} \mid \mathtt{x}, \mathtt{y} := \mathtt{w_0}, \mathtt{w_1} \models (\mathtt{x} := 0 \sqcap \mathtt{x} := 1;\ \mathtt{y} := 1) \sqcap \mathtt{x} := 1)$$
$$\longrightarrow$$
$$(\mathtt{w_0} \in \mathbb{Z} \land \mathtt{w_1} \in \mathbb{Z} \mid \mathtt{x}, \mathtt{y} := \mathtt{w_0}, \mathtt{w_1} \models \mathtt{x} := 0 \sqcap \mathtt{x} := 1;\ \mathtt{y} := 1)$$
$$\longrightarrow$$
$$(\mathtt{w_0} \in \mathbb{Z} \land \mathtt{w_1} \in \mathbb{Z} \mid \mathtt{x}, \mathtt{y} := \mathtt{w_0}, \mathtt{w_1} \models \mathtt{x} := 1;\ \mathtt{y} := 1)$$
$$\stackrel{\mathtt{x:=1}}{\Longrightarrow_{L}} \qquad \text{[Rules (11) and (4)]}$$
$$(\mathtt{w_0} \in \mathbb{Z} \land \mathtt{w_1} \in \mathbb{Z} \land \mathtt{w_2} = 1 \mid \mathtt{x}, \mathtt{y} := \mathtt{w_2}, \mathtt{w_1} \models \mathtt{Skip};\ \mathtt{y} := 1)$$
$$\longrightarrow$$
$$(\mathtt{w_0} \in \mathbb{Z} \land \mathtt{w_1} \in \mathbb{Z} \land \mathtt{w_2} = 1 \mid \mathtt{x}, \mathtt{y} := \mathtt{w_2}, \mathtt{w_1} \models \mathtt{y} := 1)$$
$$\stackrel{\mathtt{y:=1}}{\Longrightarrow_{L}} \qquad \text{[Rule (4)]}$$
$$(\mathtt{w_0} \in \mathbb{Z} \land \mathtt{w_1} \in \mathbb{Z} \land \mathtt{w_2} = 1 \land \mathtt{w_3} = 1 \mid \mathtt{x}, \mathtt{y} := \mathtt{w_2}, \mathtt{w_3} \models \mathtt{Skip})$$

This justifies the following.

$$
\begin{array}{ll}
& (\mathtt{w}_0 \in \mathbb{Z} \land \mathtt{w}_1 \in \mathbb{Z} \mid \mathtt{x}, \mathtt{y} := \mathtt{w}_0, \mathtt{w}_1 \models (\mathtt{x} := 0 \sqcap \mathtt{x} := 1;\ \mathtt{y} := 1) \sqcap \mathtt{x} := 1) \\
\longrightarrow & \\
& (\mathtt{w}_0 \in \mathbb{Z} \land \mathtt{w}_1 \in \mathbb{Z} \mid \mathtt{x}, \mathtt{y} := \mathtt{w}_0, \mathtt{w}_1 \models \mathtt{x} := 0 \sqcap \mathtt{x} := 1;\ \mathtt{y} := 1) \\
\longrightarrow & \\
& (\mathtt{w}_0 \in \mathbb{Z} \land \mathtt{w}_1 \in \mathbb{Z} \mid \mathtt{x}, \mathtt{y} := \mathtt{w}_0, \mathtt{w}_1 \models \mathtt{x} := 1;\ \mathtt{y} := 1) \\
\overset{\mathtt{x}:=1}{\Longrightarrow}_{SR} & \qquad \text{[Rules (21) and (19)]} \\
& (\mathtt{w}_0 \in \mathbb{Z} \land \mathtt{w}_1 \in \mathbb{Z} \land \mathtt{w}_2 = 1 \mid \mathtt{x}, \mathtt{y} := \mathtt{w}_2, \mathtt{w}_1 \models \mathtt{y} := 1) \\
\overset{\mathtt{y}:=1}{\Longrightarrow}_{SR} & \qquad \text{[Rule (22)]} \\
& (\mathtt{w}_0 \in \mathbb{Z} \land \mathtt{w}_1 \in \mathbb{Z} \land \mathtt{w}_2 = 1 \land \mathtt{w}_3 = 1 \mid \mathtt{x}, \mathtt{y} := \mathtt{w}_2, \mathtt{w}_3 \models \mathtt{Skip})
\end{array}
$$

We cannot, however, relate the initial configuration to any other configuration using $\Longrightarrow_{SR}$. □

We define a new transition rule that allows initial silent transitions.

$$
\frac{(\mathtt{c}_1 \mid \mathtt{s}_1 \models \mathtt{A}_1) \longrightarrow^{\epsilon *} (\mathtt{c}_2 \mid \mathtt{s}_2 \models \mathtt{A}_2) \quad (\mathtt{c}_2 \mid \mathtt{s}_2 \models \mathtt{A}_2) \overset{1}{\Longrightarrow}_{SR} (\mathtt{c}_3 \mid \mathtt{s}_3 \models \mathtt{A}_3)}{(\mathtt{c}_1 \mid \mathtt{s}_1 \models \mathtt{A}_1) \overset{1}{\Longrightarrow}_{SR} (\mathtt{c}_3 \mid \mathtt{s}_3 \models \mathtt{A}_3)} \tag{23}
$$

Example 18. Now, with Rule (23), we can proceed with Example 17 to infer the following transitions.

$$
\begin{array}{ll}
& (\mathtt{w}_0 \in \mathbb{Z} \land \mathtt{w}_1 \in \mathbb{Z} \mid \mathtt{x}, \mathtt{y} := \mathtt{w}_0, \mathtt{w}_1 \models (\mathtt{x} := 0 \sqcap \mathtt{x} := 1;\ \mathtt{y} := 1) \sqcap \mathtt{x} := 1) \\
\overset{\mathtt{x}:=1}{\Longrightarrow}_{SR} & \qquad \text{[Rules (23),(21) and (19)]} \\
& (\mathtt{w}_0 \in \mathbb{Z} \land \mathtt{w}_1 \in \mathbb{Z} \land \mathtt{w}_2 = 1 \mid \mathtt{x}, \mathtt{y} := \mathtt{w}_2, \mathtt{w}_1 \models \mathtt{y} := 1) \\
\overset{\mathtt{y}:=1}{\Longrightarrow}_{SR} & \qquad \text{[Rule (22)]} \\
& (\mathtt{w}_0 \in \mathbb{Z} \land \mathtt{w}_1 \in \mathbb{Z} \land \mathtt{w}_2 = 1 \land \mathtt{w}_3 = 1 \mid \mathtt{x}, \mathtt{y} := \mathtt{w}_2, \mathtt{w}_3 \models \mathtt{Skip})
\end{array}
$$

Once the starting configuration is defined, we have a unique $\Longrightarrow_{SR}$ transition. □

3.4 Composing labels

Transitions with labels without an event cannot be (easily) observed during the execution of a system in a testing experiment. A well known solution for this issue of observability is the use of characterising traces, which identify the current state of an SUT. We, however, want to minimise the number of such transitions, and therefore we compose transitions whenever possible.

The possibility of combination of transitions is characterised by the syntactic function $\oplus$ that combines labels; it is defined below.

$$(\mathtt{g}, \mathtt{e}, \mathtt{A}_1) \oplus \mathtt{A}_2 = (\mathtt{g}, \mathtt{e}, \mathtt{A}_1;\ \mathtt{A}_2)$$

An action can lead to a change of state, so when there is an action (different

from `Skip`) in a label, we cannot move forward any of the later guards or events. Therefore, we can only compose $(\mathtt{g}, \mathtt{e}, \mathtt{A}_1)$ with a label $\mathtt{A}_2$.

A guard potentially blocks an associated event, so if there is a guard (different from `True`) and associated event in an label, we cannot move forward any later guards. Additionally, we do not combine two labels that have events (different from ϵ). Each transition should correspond to at most one observable event. So, $(\mathtt{g}_2, \mathtt{e}, \mathtt{A})$ can only be composed with a previous label if it has only a guard $\mathtt{g}_1$.

$$\mathtt{g}_1 \oplus (\mathtt{g}_2, \mathtt{e}, \mathtt{A}) = (\mathtt{g}_1 \wedge \mathtt{g}_2, \mathtt{e}, \mathtt{A})$$

In conclusion, the domain of $\oplus$ includes exactly the pairs of labels where either the second label contains only an action, or the first label contains only a guard.

To define a system whose transitions are maximal in terms of label composition as defined by $\oplus$, we first consider a transitive closure for $\Longrightarrow_{SR}$ based on label composition. Afterwards, we define the definitive specification-oriented relation $\Longrightarrow$ as that for which no further label compositions are possible.

We define closure of $\Longrightarrow_{SR}$ in the standard way. The first rule allows a single $\Longrightarrow_{SR}$ transition to be included in the closure.

$$\frac{(\mathtt{c}_1 \mid \mathtt{s}_1 \models \mathtt{A}_1) \overset{\mathtt{l}}{\Longrightarrow}_{SR} (\mathtt{c}_2 \mid \mathtt{s}_2 \models \mathtt{A}_2)}{(\mathtt{c}_1 \mid \mathtt{s}_1 \models \mathtt{A}_1) \overset{\mathtt{l}}{\Longrightarrow}^{*}_{SR} (\mathtt{c}_2 \mid \mathtt{s}_2 \models \mathtt{A}_2)} \tag{24}$$

The second rule allows proper composition when there are two consecutive transitions with labels that can be combined according to $\oplus$.

$$\frac{(\mathtt{c}_1 \mid \mathtt{s}_1 \models \mathtt{A}_1) \overset{\mathtt{l}_1}{\Longrightarrow}^{*}_{SR} (\mathtt{c}_2 \mid \mathtt{s}_2 \models \mathtt{A}_2) \quad (\mathtt{c}_2 \mid \mathtt{s}_2 \models \mathtt{A}_2) \overset{\mathtt{l}_2}{\Longrightarrow}_{SR} (\mathtt{c}_3 \mid \mathtt{s}_3 \models \mathtt{A}_3) \quad (\mathtt{l}_1, \mathtt{l}_2) \in \operatorname{dom} \oplus}{(\mathtt{c}_1 \mid \mathtt{s}_1 \models \mathtt{A}_1) \overset{\mathtt{l}_1 \oplus \mathtt{l}_2}{\Longrightarrow}^{*}_{SR} (\mathtt{c}_3 \mid \mathtt{s}_3 \models \mathtt{A}_3)} \tag{25}$$

Our last rule defines that a $\Longrightarrow$ transition exists when there is a corresponding $\Longrightarrow^{*}_{SR}$, and it is (right) maximal, in the sense that there is no further $\Longrightarrow_{SR}$ transition from the target configuration.

$$\frac{(\mathtt{c}_1 \mid \mathtt{s}_1 \models \mathtt{A}_1) \overset{\mathtt{l}}{\Longrightarrow}^{*}_{SR} (\mathtt{c}_2 \mid \mathtt{s}_2 \models \mathtt{A}_2) \quad (\mathtt{c}_2 \mid \mathtt{s}_2 \models \mathtt{A}_2) \not\Rightarrow_{SR}}{(\mathtt{c}_1 \mid \mathtt{s}_1 \models \mathtt{A}_1) \overset{\mathtt{l}}{\Longrightarrow} (\mathtt{c}_2 \mid \mathtt{s}_2 \models \mathtt{A}_2)} \tag{26}$$

We use $(\mathtt{c}_1 \mid \mathtt{s}_1 \models \mathtt{A}_1) \not\Rightarrow_{SR}$ as an abbreviation for

$$\neg \; \exists \, \mathtt{c}_2, \mathtt{s}_2, \mathtt{A}_2, \mathtt{l} \bullet (\mathtt{c}_1 \mid \mathtt{s}_1 \models \mathtt{A}_1) \overset{\mathtt{l}}{\Longrightarrow}_{\mathtt{SR}} (\mathtt{c}_2 \mid \mathtt{s}_2 \models \mathtt{A}_2)$$

Example 19. Following from Example 16, we can use the rules above to justify

the following transitions for our example action E. Again, we present separately the two paths arising from the internal choice.

$$(\mathtt{w}_0 \in \mathbb{Z} \land \mathtt{w}_1 \in \mathbb{Z} \mid \mathtt{x}, \mathtt{y} := \mathtt{w}_0, \mathtt{w}_1 \models \mathtt{E})$$

$$\stackrel{\mathtt{x:=2}}{\Longrightarrow}$$

$$\left(\begin{array}{l} \mathtt{w}_0 \in \mathbb{Z} \land \mathtt{w}_1 \in \mathbb{Z} \land \mathtt{w}_2 = 2 \mid \mathtt{x}, \mathtt{y} := \mathtt{w}_2, \mathtt{w}_1 \\ \models \\ (\mathtt{y} > \mathtt{x} \,\&\, \mathtt{out!}(\mathtt{y} - \mathtt{x}) \rightarrow \mathtt{Skip});\ \mathtt{x} := \mathtt{y} \end{array}\right)$$

$$\stackrel{(\mathtt{y}>\mathtt{x},\mathtt{out!}(\mathtt{y}-\mathtt{x}),\mathtt{x}:=\mathtt{y})}{\Longrightarrow}$$

$$\left(\begin{array}{l} \mathtt{w}_0 \in \mathbb{Z} \land \mathtt{w}_1 \in \mathbb{Z} \land \mathtt{w}_2 = 2 \land \mathtt{w}_1 > \mathtt{w}_2 \land \mathtt{w}_3 = \mathtt{w}_1 - \mathtt{w}_2 \land \mathtt{w}_4 = \mathtt{w}_2 \mid \mathtt{x}, \mathtt{y} := \mathtt{w}_4, \mathtt{w}_1 \\ \models \\ \mathtt{Skip} \end{array}\right)$$

For the second option of the internal choice, we do not have opportunities for composition.

$$(\mathtt{w}_0 \in \mathbb{Z} \land \mathtt{w}_1 \in \mathbb{Z} \mid \mathtt{x}, \mathtt{y} := \mathtt{w}_0, \mathtt{w}_1 \models \mathtt{E})$$

$$\stackrel{\mathtt{x:=2}}{\Longrightarrow}$$

$$\left(\begin{array}{l} \mathtt{w}_0 \in \mathbb{Z} \land \mathtt{w}_1 \in \mathbb{Z} \land \mathtt{w}_2 = 2 \mid \mathtt{x}, \mathtt{y} := \mathtt{w}_2, \mathtt{w}_1 \\ \models \\ (\mathtt{inp?z} \rightarrow \mathtt{Stop});\ \mathtt{x} := \mathtt{y} \end{array}\right)$$

$$\stackrel{\mathtt{inp?z}}{\Longrightarrow}$$

$$\left(\begin{array}{l} \mathtt{w}_0 \in \mathbb{Z} \land \mathtt{w}_1 \in \mathbb{Z} \land \mathtt{w}_2 = 2 \land \mathtt{w}_3 \in \mathbb{Z} \mid \mathtt{x}, \mathtt{y} := \mathtt{w}_2, \mathtt{w}_1;\ \mathtt{var\,z} := \mathtt{w}_3 \\ \models \\ (\mathtt{let\ z} \bullet \mathtt{Stop});\ \mathtt{x} := \mathtt{y} \end{array}\right)$$

From here, we cannot proceed once again. □

All the transition relations above can be defined in the UTP *Circus* theory, so that the soundness of the transitions rules that we have defined can be formally justified. Before discussing soundness, however, we illustrate how the new transition system can be useful in practical testing techniques.

4 Test-selection criteria based on the new transition system: examples

We perceive two approaches for selection of finite test sets from a *Circus* specification. The first defines subsets of the exhaustive test sets as defined in [6] (see Section 2.3), and the second is guided by the text of the *Circus* specification. The first one is directly based on the operational semantics of *Circus*. The second one is the main motivation for the definition of the specification-oriented transition system presented above. This is what we consider in the sequel.

The selection approaches based on the structure of the tests in the exhaustive test set does not take into account the structure of the specification and the internal state changes that may occur during some unlabelled transitions of the operational semantics. The symbolic exhaustive test sets cover by construction the constrained symbolic traces of the specification. Introducing selection criteria among the constrained symbolic traces to characterise a finite subset has the merit of simplicity and of closeness to the underlying semantic model of *Circus*. However, it is the coverage of this model that is considered, and the coverage of the original specification is not taken into account.

For instance, coming back to action E of Example 1, we can note that there is no mention of the variable x and of its definition in the constrained symbolic traces. Thus, a selection criteria based on such traces cannot take into account the coverage of, for example, variable definitions and their uses.

It is the same when an operation specified by a Z schema is used in the specification: from Rule (3) of the operational semantics (see Appendix A), we can see that the associated symbolic traces does not mention the operation, and it is impossible to know which case has been covered or not by a symbolic test. Since the labels of the specification-oriented transition system contain parts of the text of the specification and record changes of state (see, for instance, Rules (3) and (4) in Section 3.2), it becomes possible to select traces (of the specification-oriented transition system, with these new labels) on the basis of the structure of the specification. For illustration, we sketch how we can use the new transition system to define data-flow-oriented test selection methods.

In the early nineties, some approaches have been proposed for generating test cases from specifications written in languages including processes interactions and data types, such as Full LOTOS, SDL, or more generally EFSM (Extended Finite State Machines). Several of these works have considered data-flow-oriented selection criteria [34, 30, 31, 27] like we do here.

Briefly, data-flow coverage criteria were originally developed for sequential imperative languages, with the coverage of definition-use associations as motivation [13]. In a data-flow graph, a definition-use association is a triple $\langle d, u, v\rangle$ where d is a node in which the variable v is defined, u is a node in which the value of v is used, and there is a definition-clear path with respect to v from d to u. The strongest data-flow criterion, all definition-use paths, requires that, for each variable, every definition-clear path (with at most one iteration by loop) is executed by a test. In order to reduce the number of paths required, weaker strategies such as all-definitions and all-uses have been defined.

When using these criteria, is is assumed that the data-flow graph has unique start and end nodes, and that there is no data-flow anomaly, that is, every path from the start node to a use of v passes through a node with a definition of v. Thus data-flow analysis is required both for checking the absence of anomalies and constructing the set of definition-use associations. (Such analysis always provide an over-approximation of data-flow dependencies due to feasibility issues).

The transposition of these criteria to the specification-oriented transition system of *Circus* requires a few adjustments. Since the relevant information is

carried by the labels of the transitions, the definition-use associations are defined as triples of two transitions and one variable. In the first transition label, the variable is defined by an assignment, or an input, or its declaration, or a Z operation where it is an output, or a specification statement in which it is in the frame. In the second transition label, it is used in a guard, or in the right-hand side of an assignment, or in an output, or in a Z operation where it is an input, or in a specification statement where it is used without decoration (in the pre or postcondition). The notion of trace is used instead of path.

Example 20. In the case of our example action E, we have a definition-use association for x whose first component is the following transition.

$$(\mathtt{w}_0 \in \mathbb{Z} \land \mathtt{w}_1 \in \mathbb{Z} \mid \mathtt{x}, \mathtt{y} := \mathtt{w}_0, \mathtt{w}_1 \models \mathtt{x} := 2)$$
$$\overset{\mathtt{x:=2}}{\Longrightarrow}$$
$$(\mathtt{w}_0 \in \mathbb{Z} \land \mathtt{w}_1 \in \mathbb{Z} \land \mathtt{w}_2 = 2 \mid \mathtt{x}, \mathtt{y} := \mathtt{w}_2, \mathtt{w}_1 \models \mathtt{Skip})$$

Indeed, x is defined in the label of this transition by an assignment. The second transition of the association is as follows.

$$\begin{pmatrix} \mathtt{w}_0 \in \mathbb{Z} \land \mathtt{w}_1 \in \mathbb{Z} \land \mathtt{w}_2 = 2 \mid \mathtt{x}, \mathtt{y} := \mathtt{w}_2, \mathtt{w}_1 \\ \models \\ (\mathtt{y} > \mathtt{x} \,\&\, \mathtt{out!}(\mathtt{y} - \mathtt{x}) \to \mathtt{Skip});\ \mathtt{x} := \mathtt{y} \end{pmatrix}$$
$$\overset{(\mathtt{y>x},\mathtt{out!}(\mathtt{y-x}),\mathtt{x:=y})}{\Longrightarrow}$$
$$\begin{pmatrix} \mathtt{w}_0 \in \mathbb{Z} \land \mathtt{w}_1 \in \mathbb{Z} \land \mathtt{w}_2 = 2 \land \mathtt{w}_1 > \mathtt{w}_2 \land \mathtt{w}_3 = \mathtt{w}_1 - \mathtt{w}_2 \land \mathtt{w}_4 = \mathtt{w}_2 \mid \mathtt{x}, \mathtt{y} := \mathtt{w}_4, \mathtt{w}_1 \\ \models \\ \mathtt{Skip} \end{pmatrix}$$

The variable x is used (twice) in this second transition, and there is an empty trace between the two transitions that is obviously definition clear with respect to x. The third component of the association is just x itself. □

Since the association in the above example is the only definition-use association for x in the simple action E, it means that it is sufficient to cover its two transitions to satisfy the criterion "all definition-use traces" for x. We note that the second definition of x in this example, namely $x := y$, does not need to be covered. It comes from the fact that it is not associated to any use. It is not a problem: since it has no effect, it would be useless to test it.

Example 21. Among the definition-use associations of action *PA* in Example 2, with respect to the local variable x_l corresponding to the program variable x, there is one whose first transition is as follows (*cf.* Examples 11 and 14).

$$\begin{pmatrix} \mathtt{w}_0 \in \mathbb{Z} \land \mathtt{w}_1 = 2 \mid \mathtt{x} := \mathtt{w}_1 \\ \models \\ \begin{pmatrix} (\mathtt{inpA?y} \to \mathtt{int!y} \to \mathtt{out!}(\mathtt{y} - \mathtt{x}) \to \mathtt{Skip}) \\ \quad [\![\{\mathtt{x}\} \mid \{\!| \mathtt{int} |\!\} \mid \{\}]\!] \\ (\mathtt{inpB?z}_1 \to \mathtt{int?z}_2 \to \mathtt{z}_1 > \mathtt{z}_2 \,\&\, \mathtt{out!}(\mathtt{z}_1 - \mathtt{x}) \to \mathtt{Skip}) \end{pmatrix} \end{pmatrix}$$

$$\overset{\mathtt{var}\,x_l,x_r:=x,x}{\Longrightarrow}$$

$$\left(\begin{array}{l} w_0 \in \mathbb{Z} \land w_1 = 2 \mid x := w_1;\ \mathtt{var}\,x_l, x_r := x, x \\ \models \\ \left(\begin{array}{l} (\mathtt{spar}\ x \mid x_l \mid x_r \mid x := x_l \bullet \mathtt{inpA?y} \rightarrow \mathtt{int!y} \rightarrow \mathtt{out!}(y - x_l) \rightarrow \mathtt{Skip}) \\ \qquad\qquad [\![\{\!| \mathtt{int} |\!\}]\!] \\ \left(\begin{array}{l} \mathtt{spar}\ x \mid x_r \mid x_l \mid \mathtt{Skip} \bullet \\ \quad \mathtt{inpB?}z_1 \rightarrow \mathtt{int?}z_2 \rightarrow z_1 > z_2 \,\&\, \mathtt{out!}(z_1 - x_r) \rightarrow \mathtt{Skip} \end{array}\right) \end{array}\right) \end{array}\right)$$

The second transition is as follows.

$$\left(\begin{array}{l} \left(\begin{array}{l} w_0 \in \mathbb{Z} \land w_1 = 2 \land w_2 \in \mathbb{Z} \land w_3 \in \mathbb{Z} \land \\ w_4 = w_2 \land w_5 \in \mathbb{Z} \land w_2 = w_5 \land w_6 = w_2 - w_1 \land w_3 > w_5 \land w_7 = w_3 - w_1 \end{array}\right) \\ \mid \left(\begin{array}{l} x := w_1;\ \mathtt{var}\,x_l, x_r := x, x; \\ \mathtt{var}\,y, z_1, z_2 := w_2, w_3, w_5;\ \mathtt{end}\,y;\ \mathtt{end}\,z_1, z_2 \end{array}\right) \\ \models \\ (\mathtt{spar}\ x \mid x_l \mid x_r \mid x := x_l \bullet \mathtt{Skip})\ [\![\{\!| \mathtt{int} |\!\}]\!]\ (\mathtt{spar}\ x \mid x_r \mid x_l \mid \mathtt{Skip} \bullet \mathtt{Skip}) \end{array}\right)$$

$$\overset{x:=x_l;\ \mathtt{end}\,x_l,x_r}{\Longrightarrow}$$

$$\left(\begin{array}{l} \left(\begin{array}{l} w_0 \in \mathbb{Z} \land w_1 = 2 \land w_2 \in \mathbb{Z} \land w_3 \in \mathbb{Z} \land \\ w_4 = w_2 \land w_5 \in \mathbb{Z} \land w_2 = w_5 \land w_6 = w_2 - w_1 \land w_3 > w_5 \land w_7 = w_3 - w_1 \end{array}\right) \\ \mid \left(\begin{array}{l} x := w_1;\ \mathtt{var}\,x_l, x_r := x, x; \\ \mathtt{var}\,y, z_1, z_2 := w_2, w_3, w_5;\ \mathtt{end}\,y;\ \mathtt{end}z_1, z_2;\ x := x_l;\ \mathtt{end}\,x_l, x_r \end{array}\right) \\ \models \\ \mathtt{Skip} \end{array}\right)$$

The third component is, of course, x_l. □

The definition-use association above forces, if the selection criterion used is "all definition-use traces" for x_l, the coverage all the interleavings of the parallel actions. In the case where the weaker criterion"all definitions" is used, following the pattern in [13], one interleaving only is required.

There are various conditions for applying data-flow testing methods to sequential programs that must be revisited for applying them to *Circus*. The existence of a unique end node can be relaxed using the observation in [26] that, in a reactive program, reaching again a start node is analogous to reaching the end node of a sequential program. Following variants of this principle, some algorithms for symbolic analysis of control dependencies are given in [26, 19] and used for data-flow analysis when there are several or no end nodes.

Data-flow analysis in presence of concurrency has been studied intensively. Of special interest in the context of *Circus* is the work in [19] for IOSTS (Input-Output Symbolic Transition Systems), where the main difference to *Circus* is that the state is not hidden, and there are no shared variables between concurrent processes. Another work of interest is the slicing algorithm for Promela in [21], where both shared variables and communications are taken into account.

5 Soundness

In the UTP, the transition rules of an operational semantics can be defined as theorems of the theory that characterises the corresponding relational model. For that, we define the transition relation in terms of the constructs of the theory (and refinement). This establishes the soundness of the operational semantics. It has been carried out for designs and CSP [18], and for *Circus* [36].

For the *Circus* operational semantics, it is defined that the transition relation $(\mathtt{c_1} \mid \mathtt{s_1} \models \mathtt{A_1}) \xrightarrow{\epsilon} (\mathtt{c_2} \mid \mathtt{s_2} \models \mathtt{A_2})$ holds if (a) there exists a valuation of the symbolic variables used in $\mathtt{c_1}$ and $\mathtt{c_2}$ such that c_1 and c_2 hold, and (b) for every such valuation, execution of A_1 in the state s_1 is refined by the execution of A_2 in s_2 [36]. By requiring that c_1 and c_2 hold, we avoid configurations with unsatisfiable state specifications. Refinement is required, not equality, since a transition reflects one, among the possibly many, available steps in the execution of A_1. As an example, we have the Rules 11 for internal choice in Appendix A: each transition captures just one of the possible choices.

For a labelled transition $(\mathtt{c_1} \mid \mathtt{s_1} \models \mathtt{A_1}) \xrightarrow{\mathtt{d.w_0}} (\mathtt{c_2} \mid \mathtt{s_2} \models \mathtt{A_2})$, it is required that execution of A_1 in s_1 is refined by an external choice between $d.w_0 \rightarrow A_2$ in the state s_2, and A_1 itself in the state s_1. This establishes that $d.w_0 \rightarrow A_2$ (in s_2) is one of the possible behaviours of A_1. The external choice captures the fact that $d.w_0$ may or may not be available, as the choice may be taken away by other behaviours of A_1. For example, if A_1 may also terminate, make some internal progress, or provide another external choice, these are all taken into account.

For the new specification-oriented transition system, we define the transition relations in terms of the constructs of the original UTP *Circus* theory, and also the $\longrightarrow$ relation of the operational semantics. For example, in the new transition system, we do not want to relate configurations that cannot be related by the *Circus* operational semantics. As already explained, it is not the objective of the new system to introduce transitions, but to remove and to annotate.

The definition for a transition $(\mathtt{c_1} \mid \mathtt{s_1} \models \mathtt{A_1}) \overset{(\mathtt{g},\mathtt{e},\mathtt{A})}{\Longrightarrow}_L (\mathtt{c_2} \mid \mathtt{s_2} \models \mathtt{A_2})$ requires

(a) $\mathtt{g} \neq \mathtt{True}$, or $\mathtt{e} \neq \epsilon$, or $\mathtt{A} \neq \mathtt{Skip}$.
(b) if $\mathtt{e}$ is a communication over a channel $\mathtt{d}$, there is a symbolic variable $\mathtt{w_0}$ such that $(\mathtt{c_1} \mid \mathtt{s_1} \models \mathtt{A_1}) \xrightarrow{\mathtt{d.w_0}} (\mathtt{c_2} \mid \mathtt{s_2} \models \mathtt{A_2})$;
(c) if $\mathtt{e}$ is ϵ, then $(\mathtt{c_1} \mid \mathtt{s_1} \models \mathtt{A_1}) \xrightarrow{\epsilon} (\mathtt{c_2} \mid \mathtt{s_2} \models \mathtt{A_2})$ holds;
(d) for all valuations of the symbolic variables that satisfy c_1 and c_2, the following properties hold:
  (d1) A_1 in s_1 is refined by the external choice between A_1 in s_1 itself, and $(g \,\&\, e \rightarrow A;\ A_2)$ also in s_1; and, finally
  (d2) $g \,\&\, \mathbf{var}\, var(\mathtt{e});\ A$ in s_1 is refined by the state s_2 guarded by g in the state s_1.

With (a), we guarantee that there are no silent transitions. The inequalities there are all syntactic, and this trivially holds for all transitions in Section 3.2. With (b) and (c), we check that, for all valuations that satisfy the constraints, there is a corresponding transition of the operational semantics. The condition (d1)

is similar to that used in the definition of labelled transitions of the operational semantics, which was explained above. The difference is that instead of considering the prefixing in the new state s_2, we use the label to construct the new state for A_2. That s_2 is indeed the appropriate next state is guaranteed by (d2), which requires that guarding A with g and declaring any variable implicitly declared by $\mathtt{e}$ is refined by s_2, guarded by g, all in s_1. If $\mathtt{e}$ is ϵ, then $var(\mathtt{e})$ is ϵ itself. We define that, in this case, the variable declaration is `Skip`, and so can be omitted.

In establishing the soundness of the transition rules, we also need to show that s_2 is a total assignment. In most cases, this is trivial. We leave a complete account of the soundness of our transition rules for another paper.

6 Conclusions and future work

In this paper, we have presented a novel transition system for a state-rich process algebra, *Circus*. Its existing operational semantics takes forwards the UTP ideas for an operational semantics for CSP by using symbolic variables to capture nondeterminism in the state. It is the basis of a testing theory for *Circus*. What we now present is an alternative characterisation of the evolutions of the *Circus* models that records information about the way in which data is defined and used. It is what we call a specification-oriented transition system for *Circus*.

We have briefly discussed how this new transition system can be used to specify selection criteria based on the use of data. Once the traces of the new transition system are selected, they are mapped to traces of the operational semantics, and in this way to tests for traces refinement and *conf*.

We have also sketched the soundness argument of the transition system. It is based on the UTP theory for *Circus*. A detailed account is going to be the subject of another paper.

In the new transition system, we still do not record in labels the internal and external choices, or parallelisms. They are recorded, of course, in the actions of the configurations. We, therefore, based solely on traces, cannot have coverage criteria that requires, for instance, covering all actions in a parallelism. For that, we will need to take into account the actions in the configurations themselves. Internal choice is covered by the use of acceptance sets in tests for *conf*.

A first piece of future work is related to completeness. We need to prove that we have the appropriate number of transitions to cater for all possible behaviours. This can be achieved by taking the transition rules as the definition of the transition relation, and recovering the denotational semantics using them. The technique applied in the UTP requires us to use the transition relation to define a semantic function that associates programs to relations of the UTP *Circus* theory. If we can prove that the relations are those defined in the denotational semantics, we have characterised the way in which concepts of the operational and denotational semantics are related in a complete and consistent way.

In the case of the our specification-based transition system, we have restricted ourselves to divergence-free processes. The completeness result, therefore, necessarily has to be qualified. The specification-oriented system is not proposed as a

replacement for the *Circus* operational semantics. For $\Longrightarrow_L$, proof of completeness can rely on the corresponding result for the operational semantics.

The most exciting plans that we have for the future, however, are the implementation of the new transition system (using a theorem prover), and the definition and application of a variety of selection criteria.

Acknowledgments

We are grateful to the Université de Paris-Sud and the Royal Academy of Engineering for their financial support of our collaboration. We would like to thank Shamim Ripon, Jim Woodcock, and Frank Zeyda for their valuable comments on the work reported here, and on a draft version of this paper.

References

1. G. Bernot, M.-C. Gaudel, and B. Marre. Software testing based on formal specifications: a theory and a tool. *Software Engineering Journal*, 6(6):387 – 405, 1991.
2. L. Bougé, N. Choquet, L. Fribourg, and M.-C. Gaudel. Test set generation from algebraic specifications using logic programming. *Journal of Systems and Software*, 6(4):343 – 360, 1986.
3. E. Brinksma. A theory for the derivation of tests. In *Protocol Specification, testing and Verification VIII*, pages 63 – 74. North-Holland, 1988.
4. A. Butterfield, A. Sherif, and J. C. P. Woodcock. Slotted Circus: A UTP-family of reactive theories. In *International Conference on Formal Enginneering*, volume 4591 of *Lecture Notes in Computer Science*, pages 75 – 97. Springer-Verlag, 2007.
5. A. L. C. Cavalcanti and M.-C. Gaudel. Testing for Refinement in CSP. In *9th International Conference on Formal Engineering Methods*, volume 4789 of *Lecture Notes in Computer Science*, pages 151 – 170. Springer-Verlag, 2007.
6. A. L. C. Cavalcanti and M.-C. Gaudel. Testing for Refinement in *Circus* – Extended version. Technical report, University of York, 2009. `www-users.cs.york.ac.uk/~alcc/CG09.pdf`.
7. A. L. C. Cavalcanti and M.-C. Gaudel. A note on traces refinement and the *conf* relation in the Unifying Theories of Programming. In A. Butterfield, editor, *Unifying Theories of Programming 2008*, volume 5713 of *Lecture Notes in Computer Science*. Springer-Verlag, 2010.
8. 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.
9. A. L. C. Cavalcanti and J. C. P. Woodcock. A Tutorial Introduction to CSP in Unifying Theories of Programming. In *Refinement Techniques in Software Engineering*, volume 3167 of *Lecture Notes in Computer Science*, pages 220 – 268. Springer-Verlag, 2006.
10. T. S. Chow. Testing Software Design Modeled by Finite-State Machines. *IEEE Transactions on Software Engineering*, SE-4(3):178 – 187, 1978.
11. J. Dick and A. Faivre. Automating the generation and sequencing of test cases from model-based specifications. In *Formal Methods Europe*, volume 670 of *Lecture Notes in Computer Science*, pages 268 – 284. Springer-Verlag, 1993.
12. E. W. Dijkstra. *A Discipline of Programming*. Prentice-Hall, 1976.

13. P. G. Frankl and E. J. Weyuker. An applicable family of data-flow testing criteria. *IEEE Transactions on Software Engineering*, 14(10):1483 – 1498, 1988.
14. J. Gannon, P. McMullin, and R. Hamlet. Data abstraction implementation, specification and testing. *ACM Transactions on Programming Languages and Systems*, 3(3):211–223, 1981.
15. M.-C. Gaudel. Testing can be formal, too. In *International Joint Conference, Theory And Practice of Software Development*, volume 915 of *Lecture Notes in Computer Science*, pages 82 – 96. Springer-Verlag, 1995.
16. W. Harwood, A. L. C. Cavalcanti, and J. C. P. Woodcock. A Theory of Pointers for the UTP. In J. S. Fitzgerald, A. E. Haxthausen, and H. Yenigun, editors, *Theoretical Aspects of Computing*, volume 5160 of *Lecture Notes in Computer Science*, pages 141 – 155. Springer-Verlag, 2008.
17. R. M. Hierons, T. H. Kim, and H. Ural. On the testability of SDL specifications. *Computer Networks*, 44(5):681–700, 2004.
18. C. A. R. Hoare and He Jifeng. *Unifying Theories of Programming*. Prentice-Hall, 1998.
19. S. Labbé and J.-P. Gallois. Slicing communicating automata specifications: polynomial algorithms for model reduction. *Formal Aspects of Computing*, 20(6):563 – 595, 2008.
20. D. Lee and M. Yannakakis. Principles and methods of testing finite state machines - A survey. In *Proceedings of the IEEE*, volume 84, pages 1090 – 1126, 1996.
21. L. I. Millett and T. Teitelbaum. Issues in slicing PROMELA and its applications to model checking, protocol understanding, and simulation. *Software Tools for Technology Transfer*, 2(4):343 – 349, 2000.
22. C. C. Morgan. *Programming from Specifications*. Prentice-Hall, 2nd edition, 1994.
23. M. V. M. Oliveira. *Formal Derivation of State-Rich Reactive Programs Using Circus*. PhD thesis, University of York, 2006.
24. M. V. M. Oliveira, A. L. C. Cavalcanti, and J. C. P. Woodcock. Unifying Theories in ProofPowerZ. *Formal Aspects of Computing, online first*, 2007. DOI 10.1007/s00165-007-0044-5.
25. M. V. M. Oliveira, A. L. C. Cavalcanti, and J. C. P. Woodcock. A UTP Semantics for *Circus*. *Formal Aspects of Computing*, 21(1-2):3 – 32, 2009.
26. V. P. Ranganath, T. Amtoft, A. Banerjee, J. Hatcliff, and M. B. Dwyer. A new foundation for control dependence and slicing for modern program structures. *ACM Transactions on Programming Languages and Systems*, 29(5):27, 2007.
27. C. Robinson-Mallett, R. M. Hierons, J. Poore, and P. Liggesmeyer. Using communication coverage criteria and partial model generation to assist software integration testing. *Software Quality Control*, 16(2):185 – 211, 2008.
28. A. W. Roscoe. *The Theory and Practice of Concurrency*. Prentice-Hall Series in Computer Science. Prentice-Hall, 1998.
29. T. L. V. L. Santos, A. L. C. Cavalcanti, and A. C. A. Sampaio. Object Orientation in the UTP. In S. Dunne and B. Stoddart, editors, *Unifying Theories of Programming*, volume 4010 of *Lecture Notes in Computer Science*, pages 18 – 37. Springer-Verlag, 2006.
30. H. V. D. Schoot and H. Ural. Data flow oriented test selection for LOTOS. *Computer Networks and ISDN Systems*, 27(7), 1993.
31. H. V. D. Schoot and H. Ural. Data flow analysis of system specifications in LOTOS. *International Journal of Software Engineering and Knowledge Engineering*, 7:43 – 68, 1997.

32. A. Sherif, A. L. C. Cavalcanti, He Jifeng, and A. C. A. Sampaio. A process algebraic framework for specification and validation of real-time systems. *Formal Aspects of Computing, online first*, 22(7):153 – 191, 2009.
33. X. Tang and J. C. P. Woodcock. Travelling Processes. In D. Kozen and C. Shankland, editors, *Mathematics of Program Construction – MPC 2004*, volume 3125 of *Lecture Notes in Computer Science*, pages 381 – 399. Springer-Verlag, 2004.
34. P. Tripathy and B. Sarikaya. Test Generation from LOTOS Specifications. *IEEE Transactions on Computers*, 40(4):543–552, 1991.
35. J. C. P. Woodcock and A. L. C. Cavalcanti. A Tutorial Introduction to Designs in Unifying Theories of Programming. In E. A. Boiten, J. Derrick, and G. Smith, editors, *IFM 2004: Integrated Formal Methods*, volume 2999 of *Lecture Notes in Computer Science*, pages 40 – 66. Springer-Verlag, 2004. Invited tutorial.
36. J. C. P. Woodcock, A. L. C. Cavalcanti, M.-C. Gaudel, and L. J. S. Freitas. Operational Semantics for *Circus*. *Formal Aspects of Computing*. To appear.
37. J. C. P. Woodcock and J. Davies. *Using Z—Specification, Refinement, and Proof*. Prentice-Hall, 1996.

A Operational semantics: table of selected transition rules

$$\frac{\mathtt{c} \wedge (\mathtt{s};\ \mathtt{p}) \wedge (\exists\, \mathtt{v}' \bullet \mathtt{s};\ \mathtt{Q})}{(\mathtt{c} \mid \mathtt{s} \models \mathtt{p} \vdash \mathtt{Q}) \xrightarrow{\epsilon} (\mathtt{c} \wedge (\mathtt{s};\ \mathtt{Q}\,[\mathtt{w}_0/\mathtt{v}']) \mid \mathtt{s};\ \mathtt{v} := \mathtt{w}_0 \models \mathtt{Skip})} \quad v' = out\alpha s \tag{1}$$

$$\frac{c}{(\mathtt{c} \mid \mathtt{s} \models \mathtt{v} := \mathtt{e}) \xrightarrow{\epsilon} (\mathtt{c} \wedge (\mathtt{s};\ \mathtt{w}_0 = \mathtt{e}) \mid \mathtt{s};\ \mathtt{v} := \mathtt{w}_0 \models \mathtt{Skip})} \tag{2}$$

$$\frac{c \wedge (s;\ \mathbf{pre}\ Op)}{(\mathtt{c} \mid \mathtt{s} \models \mathtt{Op}) \xrightarrow{\epsilon} (\mathtt{c} \wedge (\mathtt{s};\ \mathtt{Op}\,[\mathtt{w}_0/\mathtt{v}']) \mid \mathtt{s};\ \mathtt{v} := \mathtt{w}_0 \models \mathtt{Skip})} \quad v' = out\alpha s \tag{3}$$

$$\frac{c}{(\mathtt{c} \mid \mathtt{s} \models \mathtt{d!e} \rightarrow \mathtt{A}) \xrightarrow{\mathtt{d!w}_0} (\mathtt{c} \wedge (\mathtt{s};\ \mathtt{w}_0 = \mathtt{e}) \mid \mathtt{s} \models \mathtt{A})} \tag{4}$$

$$\frac{c \wedge T \neq \varnothing \qquad x \notin \alpha s}{(\mathtt{c} \mid \mathtt{s} \models \mathtt{d?x} : \mathtt{T} \rightarrow \mathtt{A}) \xrightarrow{\mathtt{d?w}_0} (\mathtt{c} \wedge \mathtt{w}_0 \in \mathtt{T} \mid \mathtt{s};\ \mathtt{var\,x} := \mathtt{w}_0 \models \mathtt{let\ x} \bullet \mathtt{A})} \tag{5}$$

$$\frac{c \wedge T \neq \varnothing \qquad x \notin \alpha s}{(\mathtt{c} \mid \mathtt{s} \models \mathtt{var\,x} : \mathtt{T} \bullet \mathtt{A}) \xrightarrow{\epsilon} (\mathtt{c} \wedge \mathtt{w}_0 \in \mathtt{T} \mid \mathtt{s};\ \mathtt{var\,x} := \mathtt{w}_0 \models \mathtt{let\ x} \bullet \mathtt{A})} \tag{6}$$

$$\frac{(\mathtt{c}_1 \mid \mathtt{s}_1 \models \mathtt{A}_1) \xrightarrow{1} (\mathtt{c}_2 \mid \mathtt{s}_2 \models \mathtt{A}_2)}{(\mathtt{c}_1 \mid \mathtt{s}_1 \models \mathtt{let\ x} \bullet \mathtt{A}_1) \xrightarrow{1} (\mathtt{c}_2 \mid \mathtt{s}_2 \models \mathtt{let\ x} \bullet \mathtt{A}_2)} \tag{7}$$

$$\frac{c}{(\mathtt{c} \mid \mathtt{s} \models \mathtt{let\ x} \bullet \mathtt{Skip}) \xrightarrow{\epsilon} (\mathtt{c} \mid \mathtt{s};\ \mathtt{end\,x} \models \mathtt{Skip})} \tag{8}$$

$$\frac{(\mathtt{c}_1 \mid \mathtt{s}_1 \models \mathtt{A}_1) \xrightarrow{1} (\mathtt{c}_2 \mid \mathtt{s}_2 \models \mathtt{A}_2)}{(\mathtt{c}_1 \mid \mathtt{s}_1 \models \mathtt{A}_1;\ \mathtt{B}) \xrightarrow{1} (\mathtt{c}_2 \mid \mathtt{s}_2 \models \mathtt{A}_2;\ \mathtt{B})} \tag{9}$$

$$\frac{c}{(\mathtt{c} \mid \mathtt{s} \models \mathtt{Skip};\ \mathtt{A}) \xrightarrow{\epsilon} (\mathtt{c} \mid \mathtt{s} \models \mathtt{A})} \tag{10}$$

$$\frac{c}{(\mathtt{c} \mid \mathtt{s} \models \mathtt{A}_1 \sqcap \mathtt{A}_2) \xrightarrow{\epsilon} (\mathtt{c} \mid \mathtt{s} \models \mathtt{A}_1)} \qquad \frac{c}{(\mathtt{c} \mid \mathtt{s} \models \mathtt{A}_1 \sqcap \mathtt{A}_2) \xrightarrow{\epsilon} (\mathtt{c} \mid \mathtt{s} \models \mathtt{A}_2)} \tag{11}$$

$$\frac{c \wedge (s;\ g)}{(\mathtt{c} \mid \mathtt{s} \models \mathtt{g} \,\&\, \mathtt{A}) \xrightarrow{\epsilon} (\mathtt{c} \wedge (\mathtt{s};\ \mathtt{g}) \mid \mathtt{s} \models \mathtt{A})} \tag{12}$$

$$\frac{c}{(\mathtt{c} \mid \mathtt{s} \models \mathtt{A}_1 \,[\![\, \mathtt{x}_1 \mid \mathtt{cs} \mid \mathtt{x}_2 \,]\!]\, \mathtt{A}_2) \xrightarrow{\epsilon} \left(\begin{array}{l} \mathtt{c} \mid \mathtt{s} \\ \models \\ (\mathtt{par\ s} \mid \mathtt{x}_1 \bullet \mathtt{A}_1) \,[\![\, \mathtt{cs} \,]\!]\, (\mathtt{par\ s} \mid \mathtt{x}_2 \bullet \mathtt{A}_2) \end{array}\right)} \tag{13}$$

$$\frac{c}{\left(\begin{array}{l} \mathtt{c} \mid \mathtt{s} \\ \models \\ \left(\begin{array}{c} (\mathtt{par\ s}_1 \mid \mathtt{x}_1 \bullet \mathtt{Skip}) \\ [\![\mathtt{cs}]\!] \\ (\mathtt{par\ s}_2 \mid \mathtt{x}_2 \bullet \mathtt{Skip}) \end{array}\right) \end{array}\right) \xrightarrow{\epsilon} (\mathtt{c} \mid (\exists\, \mathtt{x}_2' \bullet \mathtt{s}_1) \wedge (\exists\, \mathtt{x}_1' \bullet \mathtt{s}_2) \models \mathtt{Skip})} \tag{14}$$

$$\frac{(\mathtt{c} \mid \mathtt{s}_1 \models \mathtt{A}_1) \xrightarrow{\mathtt{l}} (\mathtt{c}_3 \mid \mathtt{s}_3 \models \mathtt{A}_3) \quad \mathtt{l} = \epsilon \vee \mathrm{chan}\,\mathtt{l} \notin cs}{\left(\begin{array}{l}\mathtt{c} \mid \mathtt{s} \\ \models \\ \left(\begin{array}{c}(\mathtt{par}\ \mathtt{s}_1 \mid \mathtt{x}_1 \bullet \mathtt{A}_1) \\ [\![\mathtt{cs}]\!] \\ (\mathtt{par}\ \mathtt{s}_2 \mid \mathtt{x}_2 \bullet \mathtt{A}_2)\end{array}\right)\end{array}\right) \xrightarrow{\mathtt{l}} \left(\begin{array}{l}\mathtt{c}_3 \mid \mathtt{s} \\ \models \\ \left(\begin{array}{c}(\mathtt{par}\ \mathtt{s}_3 \mid \mathtt{x}_1 \bullet \mathtt{A}_3) \\ [\![\mathtt{cs}]\!] \\ (\mathtt{par}\ \mathtt{s}_2 \mid \mathtt{x}_2 \bullet \mathtt{A}_2)\end{array}\right)\end{array}\right)} \tag{15}$$

$$\frac{(\mathtt{c} \mid \mathtt{s}_2 \models \mathtt{A}_2) \xrightarrow{\mathtt{l}} (\mathtt{c}_3 \mid \mathtt{s}_3 \models \mathtt{A}_3) \quad \mathtt{l} = \epsilon \vee \mathrm{chan}\,\mathtt{l} \notin cs}{\left(\begin{array}{l}\mathtt{c} \mid \mathtt{s} \\ \models \\ \left(\begin{array}{c}(\mathtt{par}\ \mathtt{s}_1 \mid \mathtt{x}_1 \bullet \mathtt{A}_1) \\ [\![\mathtt{cs}]\!] \\ (\mathtt{par}\ \mathtt{s}_2 \mid \mathtt{x}_2 \bullet \mathtt{A}_2)\end{array}\right)\end{array}\right) \xrightarrow{\mathtt{l}} \left(\begin{array}{l}\mathtt{c}_3 \mid \mathtt{s} \\ \models \\ \left(\begin{array}{c}(\mathtt{par}\ \mathtt{s}_1 \mid \mathtt{x}_1 \bullet \mathtt{A}_1) \\ [\![\mathtt{cs}]\!] \\ (\mathtt{par}\ \mathtt{s}_3 \mid \mathtt{x}_2 \bullet \mathtt{A}_3)\end{array}\right)\end{array}\right)} \tag{16}$$

$$\frac{\left(\begin{array}{c}(\mathtt{c} \mid \mathtt{s}_1 \models \mathtt{A}_1) \xrightarrow{\mathtt{d?w}_1} (\mathtt{c}_3 \mid \mathtt{s}_3 \models \mathtt{A}_3) \ \wedge\ (\mathtt{c} \mid \mathtt{s}_2 \models \mathtt{A}_2) \xrightarrow{\mathtt{d!w}_2} (\mathtt{c}_4 \mid \mathtt{s}_4 \models \mathtt{A}_4) \\ \vee \\ (\mathtt{c} \mid \mathtt{s}_1 \models \mathtt{A}_1) \xrightarrow{\mathtt{d!w}_1} (\mathtt{c}_3 \mid \mathtt{s}_3 \models \mathtt{A}_3) \ \wedge\ (\mathtt{c} \mid \mathtt{s}_2 \models \mathtt{A}_2) \xrightarrow{\mathtt{d?w}_2} (\mathtt{c}_4 \mid \mathtt{s}_4 \models \mathtt{A}_4) \\ \vee \\ (\mathtt{c} \mid \mathtt{s}_1 \models \mathtt{A}_1) \xrightarrow{\mathtt{d!w}_1} (\mathtt{c}_3 \mid \mathtt{s}_3 \models \mathtt{A}_3) \ \wedge\ (\mathtt{c} \mid \mathtt{s}_2 \models \mathtt{A}_2) \xrightarrow{\mathtt{d!w}_2} (\mathtt{c}_4 \mid \mathtt{s}_4 \models \mathtt{A}_4)\end{array}\right) \quad \mathtt{d} \in cs \quad c_3 \wedge c_4 \wedge w_1 = w_2}{\left(\begin{array}{l}\mathtt{c} \mid \mathtt{s} \\ \models \\ \left(\begin{array}{c}(\mathtt{par}\ \mathtt{s}_1 \mid \mathtt{x}_1 \bullet \mathtt{A}_1) \\ [\![\mathtt{cs}]\!] \\ (\mathtt{par}\ \mathtt{s}_2 \mid \mathtt{x}_2 \bullet \mathtt{A}_2)\end{array}\right)\end{array}\right) \xrightarrow{\mathtt{d!w}_2} \left(\begin{array}{l}\mathtt{c}_3 \wedge \mathtt{c}_4 \wedge \mathtt{w}_1 = \mathtt{w}_2 \mid \mathtt{s} \\ \models \\ \left(\begin{array}{c}(\mathtt{par}\ \mathtt{s}_3 \mid \mathtt{x}_1 \bullet \mathtt{A}_3) \\ [\![\mathtt{cs}]\!] \\ (\mathtt{par}\ \mathtt{s}_4 \mid \mathtt{x}_2 \bullet \mathtt{A}_4)\end{array}\right)\end{array}\right)} \tag{17}$$

$$\frac{(\mathtt{c} \mid \mathtt{s}_1 \models \mathtt{A}_1) \xrightarrow{\mathtt{d?w}_1} (\mathtt{c}_3 \mid \mathtt{s}_3 \models \mathtt{A}_3) \quad (\mathtt{c} \mid \mathtt{s}_2 \models \mathtt{A}_2) \xrightarrow{\mathtt{d?w}_2} (\mathtt{c}_4 \mid \mathtt{s}_4 \models \mathtt{A}_4) \quad \mathtt{d} \in cs \quad c_3 \wedge c_4 \wedge w_1 = w_2}{\left(\begin{array}{l}\mathtt{c} \mid \mathtt{s} \\ \models \\ \left(\begin{array}{c}(\mathtt{par}\ \mathtt{s}_1 \mid \mathtt{x}_1 \bullet \mathtt{A}_1) \\ [\![\mathtt{cs}]\!] \\ (\mathtt{par}\ \mathtt{s}_2 \mid \mathtt{x}_2 \bullet \mathtt{A}_2)\end{array}\right)\end{array}\right) \xrightarrow{\mathtt{d?w}_2} \left(\begin{array}{l}\mathtt{c}_3 \wedge \mathtt{c}_4 \wedge \mathtt{w}_1 = \mathtt{w}_2 \mid \mathtt{s} \\ \models \\ \left(\begin{array}{c}(\mathtt{par}\ \mathtt{s}_3 \mid \mathtt{x}_1 \bullet \mathtt{A}_3) \\ [\![\mathtt{cs}]\!] \\ (\mathtt{par}\ \mathtt{s}_4 \mid \mathtt{x}_2 \bullet \mathtt{A}_4)\end{array}\right)\end{array}\right)} \tag{18}$$

$$\frac{(\mathtt{c}_1 \mid \mathtt{s}_1 \models \mathtt{A}_1) \xrightarrow{\mathtt{l}} (\mathtt{c}_2 \mid \mathtt{s}_2 \models \mathtt{A}_2) \quad \mathtt{l} \neq \epsilon \quad \text{chan } \mathtt{l} \notin cs}{(\mathtt{c}_1 \mid \mathtt{s}_1 \models \mathtt{A}_1 \setminus \mathtt{cs}) \xrightarrow{\mathtt{l}} (\mathtt{c}_2 \mid \mathtt{s}_2 \models \mathtt{A}_2 \setminus \mathtt{cs})} \tag{19}$$

$$\frac{(\mathtt{c}_1 \mid \mathtt{s}_1 \models \mathtt{A}_1) \xrightarrow{\mathtt{l}} (\mathtt{c}_2 \mid \mathtt{s}_2 \models \mathtt{A}_2) \quad \mathtt{l} = \epsilon \vee \text{chan } \mathtt{l} \in cs}{(\mathtt{c}_1 \mid \mathtt{s}_1 \models \mathtt{A}_1 \setminus \mathtt{cs}) \xrightarrow{\epsilon} (\mathtt{c}_2 \mid \mathtt{s}_2 \models \mathtt{A}_2 \setminus \mathtt{cs})} \tag{20}$$

$$\frac{c}{(\mathtt{c} \mid \mathtt{s} \models \mathtt{Skip} \setminus \mathtt{cs}) \xrightarrow{\epsilon} (\mathtt{c} \mid \mathtt{s} \models \mathtt{Skip})} \tag{21}$$