RAPPORT DE RECHERCHE

L R I

DATA-FLOW COVERAGE FOR TESTING IN CIRCUS

CAVALCANTI A / GAUDEL M C

Unité Mixte de Recherche 8623
CNRS-Université Paris Sud – LRI

12/2013

Rapport de Recherche N° 1567

CNRS – Université de Paris Sud
Centre d'Orsay
LABORATOIRE DE RECHERCHE EN INFORMATIQUE
Bâtiment 650
91405 ORSAY Cedex (France)

Data-flow 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

revised version 12/2013

Abstract. *Circus* is a state-rich process algebra for refinement based on Z and CSP. In previous work, we have defined a testing theory for *Circus*, and some selection criteria based on its exhaustive test set. Here, we consider a different class of criteria, based on the text of the models, rather than directly on their operational semantics. In particular, we consider data-flow based coverage. In adapting the classical results on coverage of programs to abstract *Circus* models, we define a notion of specification traces, consider models with data anomalies, and cater for the internal nature of state and state changes. Our main results are a framework for data-flow based coverage, a novel criterion suited to state-rich process models, and a notion of instantiation of traces.

Abstract. *Circus* est une algèbre de processus avec une notion d'état interne, qui combine le pouvoir de Z pour modéliser des types abstraits de données et les constructions de CSP pour spécifier des comportements réactifs et concurrents. *Circus* permet aussi de décrire des raffinements vers des modèles concrets ou même des programmes. Les modèles abstraits impliquent couramment du nondéterminisme, qui peut venir des opérations sur les données ou des choix internes de comportement, du fait qu'on ignore les détails de l'implémentation.
Précédemment, nous avons établi une théorie du test pour *Circus*. Cette théorie est de nature symbolique : pour capturer la logique des types de données et les comportements conditionnels (gardes), nous utilisons une notion de "trace symbolique contrainte", directement dérivée de la sémantique opérationnelle du langage. Cette notion sert de base à la définition de tests symboliques associés à un modèle. Une notion d'instantiation définit comment obtenir des tests concrets.
Du fait que cette approche est conduite par la sémantique opérationnelle du langage, elle a permis de définir des jeux de tests exhaustifs et de prouver cette exhaustivité. Dans ce cadre, nous avons défini plusieurs critères de sélection de sous-ensembles des jeux de tests exhaustifs comme la couverture des traces symboliques contraintes bornées, ou la couverture des synchronisations.
Dans ce rapport, nous considérons une classe de critères de sélection de tests basés sur le texte du modèle *Circus* plutôt que sur sa sémantique opérationnelle, plus spécifiquement la couverture des flots de données.

Nous nous donnons une notion de “trace de spécification” qui collecte, en plus des évènements de communication, des opérations internes sur les données et des conditions (gardes). Sur la base de ces traces, nous formalisons les notions de définitions, usages, et sous-chemins sans définition pour *Circus*. Pour illustrer l'application de ce cadre, nous donnons les définitions des critères classiques de couverture basés dur le flot de données (all-defs, all-uses, et all-du-paths) transposés à la sélection de traces de spécification. De plus, nous formalisons un nouveau critère, mieux adapté à *Circus*, qui prend en compte les flots de données internes. Enfin, nous montrons comment construire des traces symboliques contraintes à partir des traces de spécification, et donc les tests symboliques correspondants. Les traces de spécification définies dans ce rapport peuvent être utilisées pour d'autres critères de sélection, qu'ils soient basés ou non sur les flots de données, car elles prennent en compte l'essentiel de la structure des modèles.

1 Introduction

This report presents a framework for data-flow based test selection [18] from *Circus* models; we revisit some classical criteria for coverage, and present a novel criteria especially suited for *Circus* models.

Circus [4] is a state-rich process algebra that combines freely the power of Z [23] to model abstract data types and their operations, and the CSP [20] constructs to specify reactive behaviour. As such, *Circus* is a process algebra for refinement. Nondeterminism is common in abstract models and arises both from data operations and from internal choices in patterns of interaction.

Data-flow coverage in the context of *Circus* requires adjustments. Firstly, due to the rich predicative data language of *Circus*, a concrete flow graph is likely much too big to be explicitly considered. Thus, the tests are not based on paths of a flow graph, but on specification traces. Second, data-flow anomalies must be accepted, because repeated definitions and definitions without use are routinely used in *Circus* abstract models. Finally, the state of a *Circus* process is hidden, and so not all definitions and uses, and, therefore, not all data flows, are visible.

A notable feature of the *Circus* testing theory [3] is its symbolic nature. To capture the predicative data models and guards, we have the notion of constrained symbolic traces and corresponding symbolic tests. An additional notion of instantiation defines how we can obtain concrete tests. It is in this symbolic setting that we consider here coverage based on data flows.

We state the notion of specification traces, which include, besides communication events, internal data operations and guards. Based on these traces, we formalise notions of definitions, uses, and definition-clear paths for *Circus*. To illustrate the use of this framework, we define the conventional data-flow coverage criteria all-defs, all-uses, and all-du-paths to select specification traces. In addition, we formalise a novel criteria inspired by [21] to cater for internal data flows. Finally, we consider how to construct constrained symbolic traces, and thus, corresponding symbolic tests from the specification traces. This is relevant for all selection criteria based on specification traces (and not only data-flow criteria). Given the formal setting of our work, based on the operational semantics of *Circus* and additional transition systems in [2], we can prove unbias of the selected tests. This means that they cannot reject correct systems.

In the next section, we give an overview of the notations and definitions used in our work. Section 3 presents our framework and Section 4 our new criterion. Section 5 addresses the general issue of constructing tests from selected specification traces. Finally, we consider related works in Section 6 and conclude in Section 7, where we also indicate lines for further work.

2 Background material

This section describes *Circus*, its operational semantics, and data-flow coverage.

2.1 *Circus* notation

A *Circus* model defines channels and processes like in CSP. Figure 1 presents an extract from the model of a cash machine. It uses a given set $CARD$ of valid cards, a set $Note$ of the kinds of notes available (10, 20, and 50), and a set $Cash == \mathrm{bag}\, Note$ to represent cash. The definitions of these sets are omitted.

The first paragraph in Figure 1 declares four channels: inc is used to request the withdrawal using a card of some cash, $outc$ to return a card, $cash$ to provide cash, and $refill$ to refill the note bank in the machine. The second paragraph is an explicit definition for a process called $CashMachine$.

The first paragraph of the $CashMachine$ definition is a Z schema $CMState$ marked as the **state** definition. *Circus* processes have a private state, and interact with each other and their environment using channels. The state of $CashMachine$ includes just one component: $nBank$, which is a function that records the available number of notes of each type: at most cap.

State operations can be defined by Z schemas. For instance, $DispenseNotes$ specifies an operation that takes an amount $a?$ of money as input, and outputs a bag $notes!$ of $Note$s, if there are enough available to make up the required amount. $DispenseNotes$ includes the schema $\Delta CMState$ to bring into scope the names of the state components defined in $CMState$ and their dashed counterparts to represent the state after the execution of $DispenseNotes$. To specify $notes!$, we require that the sum of its elements ($\Sigma\, notes!$) is $a?$, and that, for each kind n of $Note$, the number of notes in $notes!$ is available in the bank. $DispenseNotes$ also updates $nBank$, by decreasing its number of notes accordingly.

Another schema $DispenseError$ defines the behaviour of the operation when there are not enough notes in the bank to provide the requested amount $a?$; the result is the empty bag $[\![\ ]\!]$. The Z schema calculus is used to define the total operation $Dispense$ as the disjunction of $DispenseNotes$ and $DispenseError$.

State operations are called actions in *Circus*, and can also be defined using Morgan's specification statements [14] or guarded commands from Dijkstra's language [7]. CSP constructs can also be used to specify actions.

$[CARD]$
channel $inc : CARD \times \mathbb{N}_1; \quad outc : CARD; \; cash : Cash; \; refill$
$Note == \{10, 20, 50\}$
$Cash == \text{bag}\, Note$

process $CashMachine \mathrel{\widehat=}$ **begin**

state $CMState == [\, nBank : Note \to 0 \mathinner{..} cap \,]$

$$\begin{array}{|l}
\multicolumn{1}{l}{\underline{\; DispenseNotes \;}}\\
\Delta CMState \\
a? : \mathbb{N}_1 \\
notes! : Cash \\
\hline
\Sigma\, notes! = a? \\
\forall\, n : Note \bullet (notes! \sharp\, n) \leq nBank\, n \wedge nBank'\, n = (nBank\, n) - (notes! \sharp\, n) \\
\hline
\end{array}$$

$$\begin{array}{|l}
\multicolumn{1}{l}{\underline{\; DispenseError \;}}\\
\Xi CMState \\
a? : \mathbb{N}_1; \; notes! : Cash \\
\hline
\neg\, \exists\, ns : Cash \bullet \Sigma\, ns = a? \wedge \forall\, n : Note \bullet (ns \sharp\, n) \leq nBank\, n \\
notes! = [\![\;]\!] \\
\hline
\end{array}$$

$Dispense == DispenseNotes \vee DispenseError$

$$\bullet \left(\mu\, X \bullet \left(\begin{array}{l}
inc?c?a \to \\
\qquad X \\
\qquad \sqcap \\
\qquad outc!c \to X \\
\qquad \sqcap \\
\qquad \left(\begin{array}{l}
\mathbf{var}\; notes : Cash \bullet \\
\quad Dispense; \\
\quad \left(\begin{array}{l}
(notes \neq [\![\;]\!]) \;\&\; cash!notes \to \mathbf{Skip} \\
\Box \\
(notes = [\![\;]\!]) \;\&\; \mathbf{Skip}
\end{array} \right)
\end{array} \right) ; outc!c \to X \\
\Box \\
refill \to (nBank := \{\, 10 \mapsto cap, 20 \mapsto cap, 50 \mapsto cap \,\} \,; \; X
\end{array} \right) \right)$$

end

Fig. 1. Cash machine model

For instance, the behaviour of the process *CashMachine* is defined by a recursive action at the end after the '$\bullet$'. A recursion $\mu\ X \bullet F(X)$ has a body given by $F(X)$, where occurrences of X are recursive calls. In our example, the recursion first offers a choice between an input $inc?c?a$, which accepts a card c and a request to withdraw the amount a, and a synchronisation on *refill*, which is a request to fill the *nBank*. The actions that offer these communications are combined in an external choice ($\Box$) to be exercised by the environment.

If *refill* is chosen, an assignment changes the value of *nBank* to record a number *cap* of notes of all kinds. If $inc?c?a$ is chosen, then we have an internal (nondeterministic) choice of possible follow-on actions: recursing immediately (without returning the card or producing the money), returning the card via an output $outc!c$ before recursing, or considering the dispensation of cash before returning the card and recursing. In the dispensation, a local variable *notes* is declared, the operation *Dispense* is called, and then an external choice of two guarded actions is offered. If there is some cash available ($notes \neq [\![\ ]\!]$), then it can dispensed via $cash!notes$. Otherwise the action terminates (**Skip**). Here, nondeterminism comes from the fact that the specification does not go into details of bank management (stolen cards, bank accounts, and so on).

This example shows how Z and CSP constructs can be intermixed freely. A full account of *Circus* and its semantics is given in [17]. The *Circus* operational semantics is briefly discussed and illustrated in the next section.

2.2 *Circus* operational semantics and tests

The *Circus* operational semantics [3] is distinctive in its symbolic account of state updates. As usual, it is based on a transition relation that associates configurations and a label. For processes, the configurations are processes themselves; for actions A, they are triples of the form $(\mathtt{c} \mid \mathtt{s} \models \mathtt{A})$.

The first component `c` of those triples is a constraint over symbolic variables used to define labels and the state. These are texts that denote *Circus* predicates (over symbolic variables). We use typewriter font for pieces of text. The second component `s` is a total assignment $\mathtt{x} := \mathtt{w}$ of symbolic variables `w` to all state components `x` in scope. State assignments can also include declarations and undeclarations of variables using the constructs $\mathtt{var\ x} := \mathtt{e}$ and `end x`. The state assignments define a specific value (represented by a symbolic variable) for all variables in scope. The last component of a configuration is an action `A`.

The labels are either empty, represented by ϵ, or symbolic communications of the form `c?w` or `c!w`, where `c` is a channel name and `w` is a symbolic variable that represents an input (?) or an output (!) value.

We define the notion of traces in the expected way. Due to the symbolic nature of configurations and labels, we obtain constrained symbolic traces, or *cstraces*, for short. These are pairs formed by a sequence of labels, that is, a symbolic trace, and a constraint over the symbolic variables used in the labels.

For a process $\mathtt{begin\ state}\ [\,\mathtt{x} : \mathtt{T}\,] \bullet \mathtt{A\ end}$, the cstraces over an alphabet `a` are those of its main action `A`, starting from a state in which x takes any value

w_0 constrained by $w_0 \in T$. This is the set $cstraces^a(\mathtt{w_0} \in \mathtt{T}, \mathtt{x} := \mathtt{w_0}, \mathtt{A})$ defined below using the operational semantics (transition relation $\longrightarrow$) [3].

Definition 1.

$$
\begin{array}{l}
cstraces^{\mathsf{a}}(\mathtt{begin\ state[x : T]} \bullet \mathtt{A\ end}) = \\
\quad cstraces^{\mathsf{a}}(\mathtt{w_0} \in \mathtt{T}, \mathtt{x} := \mathtt{w_0}, \mathtt{A})
\end{array}
$$

$$
\begin{array}{l}
cstraces^{\mathsf{a}}(\mathtt{c_1}, \mathtt{s_1}, \mathtt{A_1}) = \\
\quad \left\{ \begin{array}{l} \mathtt{st}, \mathtt{c_2}, \mathtt{s_2}, \mathtt{A_2} \mid \alpha\mathtt{st} \leq \mathtt{a} \wedge (\mathtt{c_1} \mid \mathtt{s_1} \models \mathtt{A_1}) \xrightarrow{\mathtt{st}} (\mathtt{c_2} \mid \mathtt{s_2} \models \mathtt{A_2}) \\ \bullet\ (\mathtt{st}, \exists (\alpha\mathtt{c_2} \setminus \alpha\mathtt{st}) \bullet \mathtt{c_2}) \end{array} \right\}
\end{array}
$$

The parameter $\mathtt{a}$ determines the alphabet of the cstraces. Symbolic variables used in the evaluation of the operational semantics to represent internal values of the state are not included in the alphabet. As said above, $\mathtt{a}$ contains variables that denote values that are visible in the observation of a process.

Example 1. Some of the cstraces of the process *CashMachine* are as follows.

$(\langle\,\rangle, \mathtt{True})$ and $(\langle\,\mathtt{refill}, \mathtt{inc}.\alpha_0.\alpha_1, \mathtt{outc}.\alpha_2\rangle, \alpha_0 \in \mathtt{CARD} \wedge \alpha_1 \in \mathbb{N}_1 \wedge \alpha_2 = \alpha_0)$

The first is the empty cstrace (empty symbolic trace with no constraint). The second records a sequence of interactions where a request for a *refill* is followed by a request for a withdraw of an amount α_1 using card α_0, followed by the return of a card α_2. The constraint captures those arising from the declaration of *inc*, namely, α_0 is a *CARD* and α_1, a positive number. It also captures the fact that the returned card is exactly that input ($\alpha_2 = \alpha_0$). □

In the *Circus* testing theory, we take the view that, in specifications, divergences are mistakes, and in programs, they are observed as deadlocks. We, therefore, consider a theory for divergence-free models and systems under test (SUT).

As usual for process-algebra, tests of the *Circus* theory are constructed from traces. The cstraces define a set of traces: those that can be obtained by instantiating the symbolic variables so as to satisfy the constraint.

Example 2. Corresponding to the empty cstrace, we have just the empty (concrete) trace $\langle\,\rangle$. On the other hand, there are infinite instantiations of the second cstrace in the previous example. For instance, we have the following traces.

$$\langle refill, inc.0.10, outc.0\rangle \quad \langle refill, inc.0.50, outc.0\rangle \quad \langle refill, inc.1.30, outc.1\rangle$$

Here, we take 0 and 1 to be values in the set $CARD$. □

Accordingly, we have symbolic tests constructed from cstraces, and a notion of instantiation to construct concrete tests involving specific data. This approach is driven by the operational semantics of the language and led to the definition of symbolic exhaustive test sets and to proofs of their exhaustivity.

We observe that cstraces capture the constraints raised by data operations and guards, but not their structure. In [2], we have defined different transition systems whose labels capture the structure of the *Circus* model, and thus makes it possible to consider coverage of this structure.

$$
\begin{array}{lll}
\mathit{Label} & ::= & \mathit{Pred} \mid \mathit{Comm} \mid \mathit{LAct} \\
\mathit{Comm} & ::= & \epsilon \mid \mathit{CName} \mid \mathit{CName}!\mathit{Exp} \mid \mathit{CName}?\mathit{VName} \mid \mathit{CName}?\mathit{VName} : \mathit{Pred} \\
\mathit{LAct} & ::= & \mathit{VName}^{*} : [\mathit{Pred}, \mathit{Pred}] \mid \mathit{Schema} \mid \mathit{VName} := \mathit{Exp} \\
& & \mid \mathtt{var}\ \mathit{VName} : \mathit{Exp} \mid \mathtt{var}\ \mathit{VName} := \mathit{Exp} \mid \mathtt{end}\ \mathit{VName}
\end{array}
$$

Fig. 2. Syntax of specification labels.

Example 3. The following is a cstrace of *CashMachine* that captures a withdraw request followed by cash dispensation.

$$
\begin{array}{l}
(\langle\, inc.\alpha_0.\alpha_1, cash.\alpha_2 \rangle, \\
\ \alpha_0 \in \mathtt{CARD} \land \alpha_1 \in \mathbb{N}_1 \land \Sigma\,\alpha_2 = \alpha_1 \land \forall\,\mathtt{n} : \mathtt{Note} \bullet (\alpha_2 \ \sharp\ \mathtt{n}) \leq \mathtt{cap})
\end{array}
$$

The constraint defines the essential properties of the cash α_2 dispensed, but not the fact that these properties are established by variable declaration followed by a schema action call, and a guarded action. □

So, while cstraces are useful for trace-selection based on constraints, they do not support selection based on the structure of the *Circus* model. To this end, in [2] we have presented a collection of transition systems whose labels are pieces of the model: guards (predicates), communications, or simple *Circus* actions. The syntactic category of *Label*s is defined in Figure 2; the sets *Pred*, *Exp*, *CName*, *VName*, and *Schema* are those of the *Circus* predicates, expressions, channel and variable names, and Z schemas [16, 1].

In this paper, we use the transition relation $\Longrightarrow_{RP}$, called just $\Longrightarrow$ here, to define a notion of specification traces, used to consider data-flow coverage criteria. This is in contrast to what is done for sequential imperative programs where data flow graphs are considered.

2.3 Data-flow coverage

Data-flow coverage criteria were originally developed for sequential imperative languages based on the notion of definition-use associations [18]. The motivation was to check, via some test, that a variable has been assigned a correct value by causing the execution of a path in a data-flow graph from the point of assignment to a point where the assigned value is used.

Definition-use associations are traditionally defined in terms of data-flow graph as triples (d, u, v), where d is a node in which the variable v is defined, that is, some value is assigned to it, 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. In this context, 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. In order to reduce the number of tests required, weaker strategies such as *all-definitions* and *all-uses* have been defined.

$$\frac{}{(\mathtt{c} \mid \mathtt{s} \models \mathtt{A}) \overset{\langle\rangle}{\Rightarrow\!\!\!\Rightarrow} (\mathtt{c} \mid \mathtt{s} \models \mathtt{A})} \qquad \frac{(\mathtt{c}_1 \mid \mathtt{s}_1 \models \mathtt{A}_1) \overset{\mathtt{l}}{\Longrightarrow} (\mathtt{c}_2 \mid \mathtt{s}_2 \models \mathtt{A}_2)}{(\mathtt{c}_1 \mid \mathtt{s}_1 \models \mathtt{A}_1) \overset{\langle \mathtt{l} \rangle}{\Rightarrow\!\!\!\Rightarrow} (\mathtt{c}_2 \mid \mathtt{s}_2 \models \mathtt{A}_2)}$$

$$\frac{(\mathtt{c}_1 \mid \mathtt{s}_1 \models \mathtt{A}_1) \overset{\mathtt{spt}_1}{\Rightarrow\!\!\!\Rightarrow} (\mathtt{c}_2 \mid \mathtt{s}_2 \models \mathtt{A}_2) \quad (\mathtt{c}_2 \mid \mathtt{s}_2 \models \mathtt{A}_2) \overset{\mathtt{spt}_2}{\Rightarrow\!\!\!\Rightarrow} (\mathtt{c}_3 \mid \mathtt{s}_3 \models \mathtt{A}_3)}{(\mathtt{c}_1 \mid \mathtt{s}_1 \models \mathtt{A}_1) \overset{\mathtt{spt}_1 \frown \mathtt{spt}_2}{\Rightarrow\!\!\!\Rightarrow} (\mathtt{c}_3 \mid \mathtt{s}_3 \models \mathtt{A}_3)}$$

Table 1. Annotated transition relation: specification traces

When using these criteria, it is often assumed that the data-flow graph has unique start and end nodes and there is no data-flow anomaly [6]. This means that on every path from the start to the end node, there is no use of a variable v not preceded by some node with a definition of v, and that after such a node, there is always some other node with a use of v. These restrictions mainly aim at facilitating the comparison of the criteria. They are acceptable for sequential imperative programs and ensure that there is always some test sets satisfying the criteria. With our definitions such anomalies just lead to empty test sets.

Abstract specifications involving concurrency and communications, however, require adjustments to the notions underlying data-flow analysis and coverage (see, for instance [21] and [11]). We discuss some of them in Section 6 and in our work, as already said, we do not assume absence of anomalies.

3 Data-flow coverage in *Circus*

Here, we define the specification traces resulting from the transition relation $\Longrightarrow$. We then state the notions of definition and use of *Circus* variables and discuss the issue of anomalies. Afterwards, we define classical coverage criteria.

3.1 Specification traces

It is straightforward to define traces (sequences) of specification labels based on $\Longrightarrow$. The transition relation $\Rightarrow\!\!\!\Rightarrow$ annotated with such traces is defined in Table 1.

Example 4. For *CashMachine*, for instance, the following traces of specification labels, as well as their prefixes, are reachable according to $\Rightarrow\!\!\!\Rightarrow$.

$$\langle \mathtt{inc?c?a}, \mathtt{outc!c}, \mathtt{inc?c?a}, \mathtt{var\ notes} \rangle$$
$$\langle \mathtt{inc?c?a}, \mathtt{var\ notes}, \mathtt{Dispense}, \mathtt{notes} \neq [\![\ ]\!], \mathtt{cash!notes}, \mathtt{outc!c}) \rangle \qquad \square$$

We need, however, to consider enriched labels that include a tag to distinguish the various occurrences of communications and actions in the *Circus* specification. This is needed because data-flow coverage criteria are based on individual definitions or uses of a given variable occurring in the specification (or program).

Example 5. In the *CashMachine*, there are two occurrences of a use `outc!c` of the c variable: one at line 4 of the main action, and one at line 8. In the traces shown in Example 4, the two occurrences of `outc!c`; they are syntactically the same, but correspond to different occurrences of this piece of syntax in the model. Since we cannot consider repeated occurrences of labels to correspond to a single definition or use of a variable, we use tags to distinguish them. □

The tag can, for instance, be related to the position of the guards, communications, or actions in the text or in the abstract syntax tree of the model.

To get tagged labels, we just need a straightforward generalisation of the definition of $\Longrightarrow$, where a label is tagged: a pair containing a label (in the sense of the description in Section 2.2) and a tag. The value of the tag can come from information in its abstract syntax tree, for example. This is akin to the type annotation in an input $\mathtt{d?x : T}$, where $\mathtt{T}$ is the type of the channel d, an information produced by the type checking of the specification. The set of tagged specification labels is $TLabel == Label \times Tag$. We take the type Tag of tags as a given set, and do not specify a particular representation of tags.

We observe in Figure 2 that we write specification statements in the form $f : [pre, post]$, where the frame f is the list of variables potentially changed by the action specified, whose pre and postconditions are given by pre and $post$. This is the form adopted in *Circus*, coming from the refinement calculus [14]. (In the operational semantics of *Circus*, we consider a specification $pre \vdash post$, because in that context the frame plays no role. Here, we keep the frame, since it readily identifies the variables defined by this piece of the specification.)

For a process P, we define the set $sptraces(P)$ of sptraces of P: specification traces whose last label is observable, that is, a non-silent communication. This excludes traces that do not lead to new tests with respect to their prefixes, because they just include extra guards or data operations whose effect does not affect a later communication (since there is no later communication).

Definition 2.

$$sptraces(\mathtt{begin\ state}[\mathtt{x : T}] \bullet \mathtt{A\ end}) = sptraces(\mathtt{w_0} \in \mathtt{T}, \mathtt{x} := \mathtt{w_0}, \mathtt{A})$$

$$sptraces(\mathtt{c_1}, \mathtt{s_1}, \mathtt{A_1}) = \{\mathtt{spt}, \mathtt{c_2}, \mathtt{s_2}, \mathtt{A_2} \mid (\mathtt{c_1} \mid \mathtt{s_1} \models \mathtt{A_1}) \overset{\mathtt{spt}}{\Rrightarrow} (\mathtt{c_2} \mid \mathtt{s_2} \models \mathtt{A_2}) \land spt \neq \langle\,\rangle \land obs(\mathrm{last}\ spt) \bullet \mathtt{spt}\}$$

where $obs(\mathtt{l}, t) \Leftrightarrow \mathtt{l} \in Comm \land \mathtt{l} \neq \epsilon$

Without loss of generality, we consider a process $\mathtt{begin\ state}[x : T] \bullet A\ \mathtt{end}$, with state components x of type T and a main action A. Its sptraces are those of $\mathtt{A}$, when considered in the state in which x has some value identified by the symbolic variable w_0, which is constrained to satisfy $w_0 \in T$. For actions A_1, the set $sptraces(\mathtt{c_1}, \mathtt{s_1}, \mathtt{A_1})$ of its sptraces from the state characterised by the assignment s_1 and constraint c_1 is defined as those that can be constructed using $\Rrightarrow$ from the configuration $(\mathtt{c_1} \mid \mathtt{s_1} \models \mathtt{A_1})$ and whose last label is observable.

Example 6. Some sptraces of *CashMachine* are as follows. (In examples, we omit tags when they are not needed, and below we distinguish the two occurrences of $\mathtt{outc!c}$ by the tags $\mathtt{tag1}$ and $\mathtt{tag2}$.)

$$\langle \mathtt{inc?c?a}, (\mathtt{outc!c}, \mathtt{tag1})\rangle \quad \langle \mathtt{inc?c?a}, (\mathtt{outc!c}, \mathtt{tag1}), \mathtt{inc?c?a}\rangle$$
$$\langle \mathtt{inc?c?a}, \mathtt{var\ notes}, \mathtt{Dispense}, \mathtt{notes} \neq [\![\ ]\!], \mathtt{cash!notes}\rangle$$
$$\langle \mathtt{inc?c?a}, \mathtt{var\ notes}, \mathtt{Dispense}, \mathtt{notes} \neq [\![\ ]\!], \mathtt{cash!notes}, (\mathtt{outc!c}, \mathtt{tag2})\rangle$$

We note that the first specification trace in Example 4 is not an sptrace. □

The conversion of sptraces to constrained symbolic traces, which it the subject of Section 5, provides a way of obtaining symbolic tests from the specification traces. In what follows, we consider data-coverage criteria to select a subset of the sptraces of a given process P. Each of the criteria are based on the notions of definitions and uses of a given variable x, which we formalise next.

3.2 Definitions and uses

In an sptrace, a definition is a tagged label, where the label is a communication or an action that may assign a new value to a *Circus* variable, that is, an input communication, a specification statement, a Z schema where some variables are written, an assignment, or a **var** declaration, which, in *Circus* causes an initialisation. Formally, the set $\mathrm{defs}(\mathtt{x}, \mathtt{P})$ of definitions of a variable x in a process P can be identified from the set of sptraces of P as follows, where we use the function $\mathrm{defs}(\mathtt{x}, \mathtt{spt})$ that characterises the definitions of x in a particular sptrace $\mathtt{spt}$.

Definition 3. $\mathrm{defs}(\mathtt{x}, \mathtt{P}) = \bigcup \{ \mathtt{spt} : \mathit{sptraces}(\mathtt{P}) \bullet \mathrm{defs}(\mathtt{x}, \mathtt{spt}) \}$

The set $\mathrm{defs}(\mathtt{x}, \mathtt{spt})$ can be specified inductively as follows.

Definition 4. $\mathrm{defs}(\mathtt{x}, \langle\,\rangle) = \varnothing$
$\mathrm{defs}(\mathtt{x}, \mathtt{tl} ^\frown \mathtt{spt}) = (\{\mathtt{tl}\} \cap \mathrm{defs}(\mathtt{x})) \cup \mathrm{defs}(\mathtt{x}, \mathtt{spt})$

The empty trace has no definitions. If the trace is a tagged label $\mathtt{tl}$ followed by the trace $\mathtt{spt}$, we include $\mathtt{tl}$ if it is a definition of x as characterised by $\mathrm{defs}(x)$. The definitions of $\mathtt{spt}$ are themselves given by $\mathrm{defs}(\mathtt{x}, \mathtt{spt})$.

The tagged labels in which x is written (defined) can be specified as follows.

Definition 5. $\mathrm{defs}(\mathtt{x}) = \{ \mathtt{tl} : \mathit{TLabel} \mid \mathtt{x} \in \mathrm{defV}(\mathtt{tl}) \}$

The set $\mathrm{defV}(\mathtt{tl})$ of such variables for a label $\mathtt{tl}$ is specified inductively. We adopt here the convention that $\mathtt{g}$ stands for an element of *Pred*, that is, a guard label, $\mathtt{d}$ for a channel name, an element of *CName*, $\mathtt{e}$ an expression, an element of *Expr*, and $\mathtt{A}$ for a list of label actions, elements of *LAct*. We use subscripts when we need more of these meta-variables. The tags play no role here, and we ignore them in the definition of defV.

Definition 6.

$$
\begin{array}{ll}
\multicolumn{2}{l}{\mathrm{defV}(\mathtt{g}) = \mathrm{defV}(\epsilon) = \mathrm{defV}(\mathtt{d}) = \mathrm{defV}(\mathtt{d!e}) = \mathrm{defV}(\mathbf{end}\,\mathtt{y}) = \varnothing} \\
\mathrm{defV}(\mathtt{d?x}, t) = \mathrm{defV}(\mathtt{d?x : c}, t) = \{\, \mathtt{x} \,\} & \mathrm{defV}(\mathtt{f} : [\mathtt{pre}, \mathtt{post}]) = \{\, \mathtt{f} \,\} \\
\mathrm{defV}(\mathtt{Op}) = \mathit{wrtV}(\mathtt{Op}) & \mathrm{defV}(\mathtt{x := e}) = \{\, \mathtt{x} \,\} \\
\mathrm{defV}(\mathbf{var}\,\mathtt{x : T}) = \{\, \mathtt{x} \,\} & \mathrm{defV}(\mathbf{var}\,\mathtt{x := e}) = \{\, \mathtt{x} \,\}
\end{array}
$$

A Morgan specification statement $f : [\mathit{pre}, \mathit{post}]$ is a pre-post specification that can only modify the variables explicitly listed in the frame f.

The set $\mathit{wrtV}(\mathtt{Op})$ of written variables of a schema $\mathtt{Op}$ is defined in [4, page 161] to include the variables that are potentially modified by the schema, and its identification is not a purely syntactic issue. This set includes the variables v in the state of Op that are not constrained by an equality $v' = v$ in Op. Following the usual over-approximation in data-flow analysis, we can take the pessimistic, but conservative, view that Op potentially writes to all variables in scope and avoid the requirement for theorem proving.

Example 7. Coming back to the *CashMachine* (and ignoring tags) we have:

$$
\begin{array}{l}
\mathrm{defs}(\mathtt{c}, \mathtt{CashMachine}) = \{\mathtt{inc?c?a}\} \\
\mathrm{defs}(\mathtt{a}, \mathtt{CashMachine}) = \{\mathtt{inc?c?a}\} \\
\mathrm{defs}(\mathtt{notes}, \mathtt{CashMachine}) = \{\mathbf{var}\ \mathtt{notes : Cash}, \mathtt{Dispense}\} \\
\mathrm{defs}(\mathtt{nBank}, \mathtt{CashMachine}) = \{\, \mathtt{Dispense}, \\
\qquad\qquad\qquad\qquad\qquad \mathtt{nBank} := \{\, 10 \mapsto \mathtt{cap}, 20 \mapsto \mathtt{cap}, 50 \mapsto \mathtt{cap} \,\}\}
\end{array}
$$

□

The notion of (externally visible) use is simpler: a tagged label with an output communication. Formally, the set e-uses$(\mathtt{x}, \mathtt{P})$ of uses of a variable x in a process P can be identified from its set of sptraces.

Definition 7. $\text{e-uses}(\mathtt{x}, \mathtt{P}) = \bigcup \{\, \mathit{spt} : \mathit{sptraces}(\mathtt{P}) \bullet \text{e-uses}(\mathtt{x}, \mathtt{spt}) \,\}$

The set e-uses$(\mathtt{x}, \mathtt{spt})$ of uses of x in a trace $\mathtt{spt}$ can be specified as follows.

Definition 8. $\text{e-uses}(\mathtt{x}, \langle\,\rangle) = \varnothing$

$$\text{e-uses}(\mathtt{x}, \mathtt{tl} \frown \mathtt{spt}) = (\{\mathtt{tl}\} \cap \text{e-uses}(\mathtt{x})) \cup \text{e-uses}(\mathtt{x}, \mathtt{spt})$$

Finally, the general notion of uses of a variable x is defined below.

Definition 9. $\text{e-uses}(\mathtt{x}) = \{\mathtt{d} : \mathit{CName};\ \mathtt{e} : \mathit{Exp};\ t : \mathit{Tag} \mid \mathtt{x} \in \mathit{FV}(\mathtt{e}) \bullet (\mathtt{d!e}, t)\}$

These are labels $(\mathtt{d!e}, t)$ where x occurs free in the expression e. We use $FV(\mathtt{e})$ to denote the set of free variables of an expression $\mathtt{e}$.

At this point, we consider e-uses, but not the classical notion of p-uses, which relates to uses in predicates and, in the context of *Circus*, are not observable. We introduce a notion of internal uses (i-uses) later on in Section 4.1. In a *Circus* model, internal uses of a variable are its occurrences in predicates (of guards and data operations, for example) and also in assigning expressions.

Example 8. We have e-uses($\mathtt{c}, \mathtt{CashMachine}$) $= \{(\mathtt{outc!c}, \mathtt{tag1}), (\mathtt{outc!c}, \mathtt{tag2})\}$ and e-uses($\mathtt{notes}, \mathtt{CashMachine}$) $= \{\mathtt{cash!notes}\}$. There are no other externally visible uses in *CashMachine*. □

We observe that a label cannot be both a definition and a use of a variable, because a use is an output communication, which does not define any variable. Besides, a label can be neither a definition nor a use (this is the case for `refill`) and then not considered for data-flow coverage.

The property clear-path($\mathtt{spt}, \mathtt{df}, \mathtt{u}, \mathtt{x}$) characterises the fact that the trace `spt` has a subsequence that starts with the label `df`, finishes with the label `u`, with no definition of the variable x. (We note that, although we consider subsequences of a trace rather than paths of a graph, for consistency with classical terminology, we use the term clear path, rather than clear subsequence, anyway.)

Definition 10.

$$\begin{aligned} \text{clear-path}(\mathtt{spt}, \mathtt{df}, \mathtt{u}, \mathtt{x}) \Leftrightarrow\ & \exists\, i : 1 \mathinner{\ldotp\ldotp} \#\, spt \bullet spt\ i = \mathtt{df} \wedge \\ & \exists\, j : (i+1) \mathinner{\ldotp\ldotp} \#\, spt \bullet spt\ j = \mathtt{u} \wedge \\ & \forall\, k : (i+1) \mathinner{\ldotp\ldotp} (j-1) \bullet spt\ k \notin \text{defs}(\mathtt{x}, \mathtt{P}) \end{aligned}$$

Example 9. We consider the sptraces below.

$$\langle \mathtt{inc?c?a}, \mathtt{var\ notes}, \mathtt{Dispense}, \mathtt{notes} = [\![\ ]\!], (\mathtt{outc!c}, \mathtt{tag2})\rangle$$
$$\langle \mathtt{inc?c?a}, \mathtt{var\ notes}, \mathtt{Dispense}, \mathtt{notes} \neq [\![\ ]\!], \mathtt{cash!notes}, (\mathtt{outc!c}, \mathtt{tag2})\rangle$$

They have a clear path from `inc?c?a` to ($\mathtt{outc!c}, \mathtt{tag2}$) with respect to `c`. □

A e-use `u` of a variable x is said to be reachable by a definition `df` of x if there is a trace `spt` such that clear-path($\mathtt{spt}, \mathtt{df}, \mathtt{u}, \mathtt{x}$).

3.3 Data-flow anomalies and *Circus*

Three data-flow anomalies are usually identified: (1) a use of a variable without a previous definition; (2) two definitions without an intermediate use; and (3) a definition without use. While these all raise concerns in a program, it is not the case of (2) and (3) in a *Circus* model. Because a variable declaration is a variable definition that assigns an arbitrary value to a variable, it is common to follow it up with a second definition that restricts that value.

In addition, it is not rare to use a communication `d?x` to define just that the value x to be input via the channel `d` is not restricted (and also later not used). In an abstract specification, a process involving such a communication might, for example, be combined in parallel with another process that captures another requirement concerned with restricting these values x, while the requirement captured by the process that defines `d?x` is not concerned with such values.

As we can see in the following sections, the data-coverage criteria that we consider are based on the set of definitions of a variable x. When a definition involved in any of the above anomalies is considered, it imposes no restriction on the set of tests under consideration for coverage. In practical terms, no tests are required as a consequence of the presence of such definitions.

3.4 all-defs

The first data-coverage criterion that we consider, all-defs, requires that all definitions are covered, and followed by one (reachable) use, via any (clear) path. We formalise coverage criterion by identifying the sets of sptraces $SSPT$ that satisfy that criterion. For all-defs, the formal definition is as follows.OK

Definition 11. *For every variable name x and process P, a set $SSPT$ of sptraces of P provides all-defs coverage if, and only if,*

$$\begin{array}{l} \forall\, \mathtt{df} : \mathrm{defs}(\mathtt{x},\mathtt{P}) \bullet \\ \quad (\exists\, \mathtt{spt} : \mathit{sptraces}(\mathtt{P});\ \mathtt{u} : \text{e-uses}(\mathtt{x},\mathtt{P}) \bullet \text{clear-path}(\mathtt{spt},\mathtt{df},\mathtt{u},\mathtt{x})) \Rightarrow \\ \quad\quad (\exists\, \mathtt{spt} : \mathit{SSPT};\ \mathtt{u} : \text{e-uses}(\mathtt{x},\mathtt{P}) \bullet \text{clear-path}(\mathtt{spt},\mathtt{df},\mathtt{u},\mathtt{x})) \end{array}$$

It requires that, if there is an sptrace `spt` that can contribute to coverage, then at least one is included in $SSPT$. Since, as already explained, data-flow anomalies are acceptable, this set may be empty.

Example 10. As explained in Examples 7 and 8, in *CashMachine*, `inc?c?a` is the only definition of c, and its two uses are (`outc!c`, `tag1`) and (`outc!c`, `tag2`). There is a trivial clear path between `inc?c?a` to (`outc!c`, `tag1`), where the use immediately follows the definition. Also, as indicated in Example 9, there are two clear paths from `inc?c?a` to (`outc!c`, `tag2`). Accordingly, examples of sets of sptraces that provide all-defs coverage are the three singletons below.

$$\begin{array}{l} \{\langle \mathtt{inc?c?a}, (\mathtt{outc!c}, \mathtt{tag1})\rangle\} \\ \{\langle \mathtt{inc?c?a}, \mathtt{var\ notes}, \mathtt{Dispense}, \mathtt{notes} = [\![\ ]\!], (\mathtt{outc!c}, \mathtt{tag2})\rangle\} \\ \{\langle \mathtt{inc?c?a}, \mathtt{var\ notes}, \mathtt{Dispense}, \mathtt{notes} \neq [\![\ ]\!], \mathtt{cash!notes}, (\mathtt{outc!c}, \mathtt{tag2})\rangle\} \end{array}$$

Other sets that provide all-defs coverage are the supersets of the above sets, and the sets that include any of the extensions of the sptraces above. □

3.5 all-uses

The all-uses criterion requires that all definition-use pairs are covered by at least one clear path, if possible.

Definition 12. *For every variable name x and process P, a set $SSPT$ of sptraces of P provides all-uses coverage if, and only if,*

$$\begin{array}{l} \forall\, \mathtt{df} : \mathrm{defs}(\mathtt{x},\mathtt{P});\ \mathtt{u} : \text{e-uses}(\mathtt{x},\mathtt{P}) \bullet \\ \quad (\exists\, \mathtt{spt} : \mathit{sptraces}(\mathtt{P}) \bullet \text{clear-path}(\mathtt{spt},\mathtt{df},\mathtt{u},\mathtt{x})) \Rightarrow \\ \quad\quad (\exists\, \mathtt{spt} : \mathit{SSPT} \bullet \text{clear-path}(\mathtt{spt},\mathtt{df},\mathtt{u},\mathtt{x})) \end{array}$$

Example 11. Sets of sptraces that provide all-uses coverage are obtained by taking the first one of Example 10 and one of the other ones.

$$\begin{array}{l} \{\,\langle \mathtt{inc?c?a}, (\mathtt{outc!c}, \mathtt{tag1})\rangle, \\ \ \ \langle \mathtt{inc?c?a}, \mathtt{var\ notes}, \mathtt{Dispense}, \mathtt{notes} = [\![\ ]\!], (\mathtt{outc!c}, \mathtt{tag2})\rangle\} \\ \{\,\langle \mathtt{inc?c?a}, (\mathtt{outc!c}, \mathtt{tag1})\rangle, \\ \ \ \langle \mathtt{inc?c?a}, \mathtt{var\ notes}, \mathtt{Dispense}, \mathtt{notes} \neq [\![\ ]\!], \mathtt{cash!notes}, (\mathtt{outc!c}, \mathtt{tag2})\rangle\} \end{array}$$

These sets have two elements since there is one definition of c, and two uses. □

3.6 all-du-paths

The all-du-paths criterion requires that all definition-use pairs are covered by all possible paths. Our notion of path, as already said, is based on sptraces.

Definition 13. *For every variable name x and process P, a set $SSPT$ of sptraces of P provides all-du-paths coverage if, and only if,*

$$\begin{array}{l}\forall\, \mathtt{df} : \mathrm{defs}(\mathtt{x}, \mathtt{P});\ \mathtt{u} : \text{e-uses}(\mathtt{x}, \mathtt{P});\ \mathtt{p} : \text{all-du-sub-path}(\mathtt{x}, \mathtt{P}, \mathtt{df}, \mathtt{u}) \bullet \\ \quad \exists\, \mathtt{spt} : SSPT;\ \mathtt{spt}_1, \mathtt{spt}_2 : \mathrm{seq}\ TLabel \bullet \mathtt{spt} = spt_1 \frown p \frown spt_2\end{array}$$

The traces $\mathtt{spt}_1$ and $\mathtt{spt}_2$ are an initialisation and a finalisation trace that determine an sptrace of P that covers $\mathtt{p}$. The set all-du-sub-path$(\mathtt{x}, \mathtt{P}, \mathtt{df}, \mathtt{u})$ contains all the paths in P, according to its set of sptraces, that start with $\mathtt{df}$, finish with $\mathtt{u}$, and is clear of definitions of x in between.

Definition 14.

$$\begin{array}{l}\text{all-du-sub-path}(\mathtt{x}, \mathtt{P}, \mathtt{df}, \mathtt{u}) = \\ \quad \{\, \mathtt{spt} : sptraces(\mathtt{P});\ i : 1 \mathinner{..} \#\, spt;\ j : (i+1) \mathinner{..} \#\, spt \mid \\ \qquad\quad spt\ i = \mathtt{df} \land spt\ j = \mathtt{u} \land \forall\, k : (i+1) \mathinner{..} (j-1) \bullet spt\ k \notin \mathrm{defs}(\mathtt{x}, \mathtt{P})) \\ \qquad \bullet\ (i \mathinner{..} j) \upharpoonright spt\, \}\end{array}$$

Example 12. A set of sptraces that provides all-du-paths coverage is obtained by selecting the three sptraces of example 10

$$\begin{array}{l}\{\, \langle \mathtt{inc?c?a}, (\mathtt{outc!c}, \mathtt{tag1}) \rangle, \\ \ \ \langle \mathtt{inc?c?a}, \mathtt{var\ notes}, \mathtt{Dispense}, \mathtt{notes} = [\![\ ]\!], (\mathtt{outc!c}, \mathtt{tag2}) \rangle \\ \ \ \langle \mathtt{inc?c?a}, \mathtt{varnotes}, \mathtt{Dispense}, \mathtt{notes} \neq [\![\ ]\!], \mathtt{cash!notes}, (\mathtt{outc!c}, \mathtt{tag2}) \rangle \}\end{array}$$

□

Example 13. When considering data-flow coverage of the variable *notes*, we observe that it is defined in the *Dispense* schema and it has one use only, $\mathtt{cash!notes}$ at line 8 of the main action. There is a unique def-clear path between them. Thus a single sptrace is sufficient to provide coverage according to any of the three criteria: it just needs to contain the two consecutive labels corresponding to this definition and this use as indicated below.

$$\langle \ldots, \mathtt{inc?c?a}, \mathtt{var\ notes}, \mathtt{Dispense}, \mathtt{notes} \neq [\![\ ]\!], \mathtt{cash!notes}, \ldots \rangle$$

For instance, the singleton below provides all-defs, all-uses and all-du-paths coverage with respect to the variable *notes*.

$$\{ \langle \mathtt{inc?c?a}, \mathtt{var\ notes}, \mathtt{Dispense}, \mathtt{notes} \neq [\![\ ]\!], \mathtt{cash!notes}, (\mathtt{outc!c}, \mathtt{tag2}) \rangle \}$$

□

This mostly concludes our discussion of the standard data-flow coverage criteria. We note, however, that the *CashMachine* variables are c, a, *notes*, and *nBank*, and that *nBank* and a are used internally only. Thus, there is no def-clear path from their definition to an external use, and given the definitions of all-defs, all-uses and all-du-paths, every set of sptraces provides coverage with respect to these criteria and these variables. They contribute, however, to our next more elaborate criterion, which takes the nature of *Circus* models into account, emphasizing dependencies between different variables.

In addition, the structure of schemas is not taken into account. For instance, *Dispense* is a disjunction, and the criteria above do not force the coverage of the two cases, even if the last one achieves it due to the existence of two definition-clear paths that cover them. Coverage of the structure of Z schemas could be another selection criterion by itself, or combined with data-flow analysis.

4 sel-var-df-chain-trace

The definition of this criterion is based on the notion of a var-df-chain, which we introduce first (Section 4.1). Afterwards, we formalise this novel criterion (Section 4.2), and lastly we apply it to the *CashMachine* (Section 4.3). Roughly, the idea is to identify sptraces that include chains of definition and associated internal uses of variables, such that each variable affects the next one in the chain. For state-rich models, we expect an interesting number of such chains.

4.1 var-df-chain

A suffix of an sptrace `spt` starting at position i (that is, $(i \mathinner{..} \# spt) \upharpoonleft spt$) is in the set var-df-chain$(\mathtt{x}, \mathtt{P})$ of var-df-chains of P for x if it starts with a label $spt\ i$ that defines x and subsequently has a clear path to a label $spt\ j$. This label must either be a use of x, and in this case it must be the last label of `spt`, or affect the definition of another variable y, and in this case `spt` must continue with a var-df-chain for `y`. The continuation is determined by $(j \mathinner{..} \# spt) \upharpoonleft spt$, the subsequence of spt from the position j.

Definition 15.

$$
\begin{array}{l}
\text{var-df-chain}(\mathtt{x},\mathtt{P}) = \\
\quad \{\ \mathtt{spt} : sptraces(P);\ i : 1 \mathinner{..} \# spt;\ j : (i+1) \mathinner{..} \# spt \mid \\
\qquad \left(\begin{array}{l}
spt\ i \in \text{defs}(\mathtt{x},\mathtt{P}) \wedge (\forall k : (i+1) \mathinner{..} (j-1) \bullet spt\ k \notin \text{defs}(\mathtt{x},\mathtt{P})) \wedge \\
\left(\begin{array}{l}
(spt\ j \in \text{e-uses}(\mathtt{x},\mathtt{P}) \wedge j = \# spt) \vee \\
(\exists\, \mathtt{y} \bullet \text{affects}(\mathtt{x},\mathtt{y}, spt\ j) \wedge (j \mathinner{..} \# spt) \upharpoonleft spt \in \text{var-df-chain}(\mathtt{y},\mathtt{P}))
\end{array}\right)
\end{array}\right) \\
\qquad \bullet\ (i \mathinner{..} \# spt) \upharpoonleft spt \\
\quad \}
\end{array}
$$

A variable x affects the definition of another variable y in a tagged label `tl` if it is an internal use of x and a definition of y.

Definition 16. $\text{affects}(\mathtt{x},\mathtt{y},\mathtt{tl}) = \mathtt{x} \in \text{i-useV}(\mathtt{tl}) \wedge \mathtt{y} \in \text{defs}(\mathtt{tl})$

An internal use of a variable is an occurrence of it in a guard or action. (This notion of internal use subsumes the classical notion of p-uses.) The set i-useV($\mathtt{tl}$) of variables used in a tagged label $\mathtt{tl}$, internally, is defined as follows.

Definition 17.

$$\begin{array}{ll}
\text{i-useV}(\mathtt{g}) = FV(\mathtt{g}) & \text{i-useV}(\epsilon) = \text{i-useV}(\mathtt{d}) = \varnothing \\
\text{i-useV}(\mathtt{d!e}) = \text{i-useV}(\mathtt{d?x}) = \varnothing & <<<<<<< .mine\,\text{i-useV}(\mathtt{d?x} : \mathtt{g}) = FV(\mathtt{g}) \setminus \{x\} ======= \text{i-useV}(
\end{array}$$

$$\text{i-useV}(\mathtt{f} : [\mathtt{pre}, \mathtt{pos}]) = FV(\mathtt{pre}) \cup FV(\mathtt{pos})$$

$$\begin{array}{ll}
\text{i-useV}(\mathtt{Op}) = FV(\mathtt{Op}) & \text{i-useV}(\mathtt{x} := \mathtt{e}) = FV(\mathtt{e}) \\
\text{i-useV}(\mathtt{var\,x} : \mathtt{T}) = \varnothing & \text{i-useV}(\mathtt{var\,x} := \mathtt{e}) = FV(\mathtt{e}) \\
\text{i-useV}(\mathtt{end\,y}) = \varnothing &
\end{array}$$

We observe that not all free occurrences of a variable constitute an internal use of it. For example, an assignment to a variable is not an use of it.

4.2 The criterion

We observe that var-df-chains are not sptraces, but suffixes of sptraces. So, coverage is provided by sptraces that have such suffixes, rather than by the var-df-chains themselves. In particular, sel-var-df-chain-trace coverage requires that every chain in a model is covered by at least one sptrace.

Definition 18. *For every variable name x and process P, a set $SSPT$ of sptraces of P provides sel-var-df-chain-trace coverage if, and only if,*

$$\begin{array}{l}
\forall\, \mathtt{spt_1} : \text{var-df-chain}(\mathtt{x}, \mathtt{P}) \bullet \\
\quad \exists\, \mathtt{spt_2} : SSPT;\ \mathtt{spt_3} : \operatorname{seq} TLabel \bullet spt_2 = spt_3 \frown spt_1
\end{array}$$

The specification trace $\mathtt{spt_3}$ is an initialisation trace that leads to the chain.

This is the most demanding of the criteria in this report as shown below.

Theorem 1 *For every set $SSPT$ of sptraces, if it provides sel-var-df-chain-trace coverage, then it provides all-du-paths coverage. Additionally, if it provides all-du-paths coverage, then it provides all-uses coverage. Finally, if it provides all-uses coverage, then it provides all-defs coverage.*

The proof of this theorem uses our detailed formalisation of all definitions. It establishes subset inclusion for each pair of the sets of sets of sptraces that provide coverage according to Definitions 11, 12, 13 and 18.

Proof.
Case sel-var-df-chain-trace ensures all-paths

$$\left(\begin{array}{l}
\forall\, \mathtt{spt_1} : \text{var-df-chain}(\mathtt{x}, \mathtt{P}) \bullet \\
\quad \exists\, \mathtt{spt_2} : SSPT;\ \mathtt{spt_3} : \operatorname{seq} TLabel \bullet spt_2 = spt_3 \frown spt_1
\end{array}\right)$$

$\Leftrightarrow$

$$\left(\begin{array}{l}
\forall \mathtt{spt_1} : \mathrm{seq}\ TLabel;\ \mathtt{spt_4} : sptraces(P); \\
\quad i : 1 \mathrel{..} \#\, spt_4;\ j : (i+1) \mathrel{..} \#\, spt_4 \bullet \\
\quad \left(\begin{array}{l}
\left(\begin{array}{l}
spt_4\, i \in \mathrm{defs}(\mathtt{x},\mathtt{P}) \land \\
(\forall k : (i+1) \mathrel{..} (j-1) \bullet spt_4\, k \notin \mathrm{defs}(\mathtt{x},\mathtt{P})) \land \\
\left(\begin{array}{l}
(spt_4\, j \in \text{e-uses}(\mathtt{x},\mathtt{P}) \land j = \#\, spt_4) \lor \\
\left(\begin{array}{l}
\exists\, \mathtt{y} \bullet \\
\qquad \mathrm{affects}(\mathtt{x},\mathtt{y},spt_4\, j) \land \\
\qquad (j \mathrel{..} \#\, spt) \upharpoonright spt_4 \in \text{var-df-chain}(\mathtt{y},\mathtt{P}))
\end{array}\right)
\end{array}\right) \land \\
spt_1 = (i \mathrel{..} \#\, spt_4) \upharpoonright spt_4
\end{array}\right) \\
\Rightarrow \\
\quad \exists\, \mathtt{spt_2} : SSPT;\ \mathtt{spt_3} : \mathrm{seq}\ TLabel \bullet spt_2 = spt_3 \frown spt_1
\end{array}\right)
\end{array}\right)$$

[definition of var-df-chain($\mathtt{x},\mathtt{P}$)]

$$\Rightarrow$$

$$\left(\begin{array}{l}
\forall \mathtt{spt_1} : \mathrm{seq}\ TLabel;\ \mathtt{spt_4} : sptraces(P); \\
\quad i : 1 \mathrel{..} \#\, spt_4;\ j : (i+1) \mathrel{..} \#\, spt_4 \bullet \\
\quad \left(\begin{array}{l}
\left(\begin{array}{l}
spt_4\, i \in \mathrm{defs}(\mathtt{x},\mathtt{P}) \land \\
(\forall k : (i+1) \mathrel{..} (j-1) \bullet spt_4\, k \notin \mathrm{defs}(\mathtt{x},\mathtt{P})) \land \\
spt_4\, j \in \text{e-uses}(\mathtt{x},\mathtt{P}) \land j = \#\, spt_4 \land \\
spt_1 = (i \mathrel{..} \#\, spt_4) \upharpoonright spt_4
\end{array}\right) \Rightarrow \\
\quad \exists\, \mathtt{spt_2} : SSPT;\ \mathtt{spt_3} : \mathrm{seq}\ TLabel \bullet spt_2 = spt_3 \frown spt_1
\end{array}\right)
\end{array}\right)$$

[predicate calculus]

$$\Leftrightarrow \left(\begin{array}{l}
\forall \mathtt{spt_4} : sptraces(P);\ i : 1 \mathrel{..} \#\, spt_4;\ \mathtt{p} : \mathrm{seq}\ TLabel \bullet \\
\qquad \left(\begin{array}{l}
\left(\begin{array}{l}
\exists\, \mathtt{df} : \mathrm{defs}(\mathtt{x},\mathtt{P});\ \mathtt{u} : \text{e-uses}(\mathtt{x},\mathtt{P}) \bullet \\
\qquad \left(\begin{array}{l}
spt_4\, i = \mathtt{df} \land spt_4\, (\#\, spt_4) = \mathtt{u} \land \\
(\forall k : (i+1) \mathrel{..} (\#\, spt_4 - 1) \bullet spt_4\, k \notin \mathrm{defs}(\mathtt{x},\mathtt{P})) \land \\
p = (i \mathrel{..} \#\, spt_4) \upharpoonright spt_4
\end{array}\right)
\end{array}\right) \\
\Rightarrow \\
\exists\, \mathtt{spt} : SSPT;\ \mathtt{spt_1} : \mathrm{seq}\ TLabel \bullet spt = spt_1 \frown p)
\end{array}\right)
\end{array}\right)$$

[property of sets]

$$\Leftrightarrow \left(\begin{array}{l}
\forall\, \mathtt{spt_4} : sptraces(P);\ i : 1 \mathrel{..} \#\, spt_4; \\
\quad \mathtt{df} : \mathrm{defs}(\mathtt{x},\mathtt{P});\ \mathtt{u} : \text{e-uses}(\mathtt{x},\mathtt{P});\ \mathtt{p} : \mathrm{seq}\ TLabel \bullet \\
\qquad \left(\begin{array}{l}
\left(\begin{array}{l}
spt_4\, i = \mathtt{df} \land spt_4\, (\#\, spt_4) = \mathtt{u} \land \\
(\forall k : (i+1) \mathrel{..} (\#\, spt_4 - 1) \bullet spt_4\, k \notin \mathrm{defs}(\mathtt{x},\mathtt{P})) \land \\
p = (i \mathrel{..} \#\, spt_4) \upharpoonright spt_4
\end{array}\right) \Rightarrow \\
\quad \exists\, \mathtt{spt} : SSPT;\ \mathtt{spt_1} : \mathrm{seq}\ TLabel \bullet spt = spt_1 \frown p
\end{array}\right)
\end{array}\right)$$

[predicate calculus]

$$\Leftrightarrow \left(\begin{array}{l}\forall\, \mathtt{spt_4} : \mathit{sptraces}(\mathtt{P});\ i : 1 \mathrel{..} \#\, \mathit{spt_4};\ j : i \mathrel{..} \#\, \mathit{spt_4};\\ \quad \mathtt{df} : \mathrm{defs}(\mathtt{x}, \mathtt{P});\ \mathtt{u} : \text{e-uses}(\mathtt{x}, \mathtt{P});\ \mathtt{p} : \mathrm{seq}\ \mathit{TLabel} \bullet\\ \quad \left(\begin{array}{l}\left(\begin{array}{l}\mathit{spt_4}\, i = \mathtt{df} \land \mathit{spt_4}\, j = \mathtt{u} \land\\ (\forall\, k : (i+1) \mathrel{..} (j-1) \bullet \mathit{spt_4}\, k \notin \mathrm{defs}(\mathtt{x}, \mathtt{P})) \land\\ p = (i \mathrel{..} j) \upharpoonright \mathit{spt_4}\end{array}\right) \Rightarrow\\ \quad (\exists\, \mathtt{spt} : \mathit{SSPT};\ \mathtt{spt_1} : \mathrm{seq}\ \mathit{TLabel} \bullet \mathtt{spt} = \mathit{spt_1} \frown p)\end{array}\right)\end{array}\right)$$

$$[\{\mathtt{spt} : \mathit{sptraces}(\mathtt{P});\ i : 1 \mathrel{..} \#\, \mathit{spt} \mid P(\mathit{spt}, \#\, \mathit{spt}) \bullet (i \mathrel{..} \#\, \mathit{spt}) \upharpoonright \mathit{spt}\} =]$$

$$[\{\mathtt{spt} : \mathit{sptraces}(\mathtt{P});\ i : 1 \mathrel{..} \#\, \mathit{spt};\ j;\ i \mathrel{..} \#\, \mathit{spt} \mid P(\mathit{spt}, j) \bullet (i \mathrel{..} j) \upharpoonright \mathit{spt}\}]$$

[since $\mathit{sptraces}(\mathtt{P})$ is prefix closed]

$$\Leftrightarrow \left(\begin{array}{l}\forall\, \mathtt{spt_4} : \mathit{sptraces}(\mathtt{P});\ i : 1 \mathrel{..} \#\, \mathit{spt_4};\ j : (i+1) \mathrel{..} \#\, \mathit{spt_4};\\ \quad \mathtt{df} : \mathrm{defs}(\mathtt{x}, \mathtt{P});\ \mathtt{u} : \text{e-uses}(\mathtt{x}, \mathtt{P});\ \mathtt{p} : \mathrm{seq}\ \mathit{TLabel} \bullet\\ \quad \left(\begin{array}{l}\left(\begin{array}{l}\mathit{spt_4}\, i = \mathtt{df} \land \mathit{spt_4}\, j = \mathtt{u} \land\\ (\forall\, k : (i+1) \mathrel{..} (j-1) \bullet \mathit{spt_4}\, k \notin \mathrm{defs}(\mathtt{x}, \mathtt{P})) \land\\ p = (i \mathrel{..} j) \upharpoonright \mathit{spt_4}\end{array}\right) \Rightarrow\\ \quad (\exists\, \mathtt{spt} : \mathit{SSPT};\ \mathtt{spt_1} : \mathrm{seq}\ \mathit{TLabel} \bullet \mathtt{spt} = \mathit{spt_1} \frown p)\end{array}\right)\end{array}\right)$$

[$\mathtt{df} \neq \mathtt{u}$ by $\mathrm{defs}(\mathtt{x}, \mathtt{P}) \cap \text{e-uses}(\mathtt{x}, \mathtt{P}) = \varnothing$]

$$\Rightarrow \left(\begin{array}{l}\forall\, \mathtt{spt_4} : \mathit{sptraces}(\mathtt{P});\ i : 1 \mathrel{..} \#\, \mathit{spt_4};\ j : (i+1) \mathrel{..} \#\, \mathit{spt_4};\\ \quad \mathtt{df} : \mathrm{defs}(\mathtt{x}, \mathtt{P});\ \mathtt{u} : \text{e-uses}(\mathtt{x}, \mathtt{P});\ \mathtt{p} : \mathrm{seq}\ \mathit{TLabel} \bullet\\ \quad \left(\begin{array}{l}\left(\begin{array}{l}\mathit{spt_4}\, i = \mathtt{df} \land \mathit{spt_4}\, j = \mathtt{u} \land\\ (\forall\, k : (i+1) \mathrel{..} (j-1) \bullet \mathit{spt_4}\, k \notin \mathrm{defs}(\mathtt{x}, \mathtt{P})) \land\\ p = (i \mathrel{..} j) \upharpoonright \mathit{spt_4}\end{array}\right) \Rightarrow\\ \quad \left(\begin{array}{l}\exists\, \mathtt{spt} : \mathit{SSPT};\ \mathtt{spt_1}, \mathtt{spt_2} : \mathrm{seq}\ \mathit{TLabel} \bullet\\ \quad \mathtt{spt} = \mathit{spt_1} \frown p \frown \mathit{spt_2}\end{array}\right)\end{array}\right)\end{array}\right)$$

[take $\mathit{spt_2} = \langle\,\rangle$]

$$\Leftrightarrow \left(\begin{array}{l}\forall\, \mathtt{df} : \mathrm{defs}(\mathtt{x}, \mathtt{P});\ \mathtt{u} : \text{e-uses}(\mathtt{x}, \mathtt{P});\ \mathtt{p} : \mathrm{seq}\ \mathit{TLabel} \bullet\\ \quad \left(\begin{array}{l}\left(\begin{array}{l}\exists\, \mathtt{spt} : \mathit{sptraces}(\mathtt{P});\ i : 1 \mathrel{..} \#\, \mathit{spt};\ j : (i+1) \mathrel{..} \#\, \mathit{spt} \bullet\\ \quad \mathit{spt}\, i = \mathtt{df} \land \mathit{spt}\, j = \mathtt{u} \land\\ \quad (\forall\, k : (i+1) \mathrel{..} (j-1) \bullet \mathit{spt}\, k \notin \mathrm{defs}(\mathtt{x}, \mathtt{P})) \land\\ \quad p = (i \mathrel{..} j) \upharpoonright \mathit{spt}\end{array}\right) \Rightarrow\\ \quad \left(\begin{array}{l}\exists\, \mathtt{spt} : \mathit{SSPT};\ \mathtt{spt_1}, \mathtt{spt_2} : \mathrm{seq}\ \mathit{TLabel} \bullet\\ \quad \mathtt{spt} = \mathit{spt_1} \frown p \frown \mathit{spt_2}\end{array}\right)\end{array}\right)\end{array}\right)$$

[predicate calculus]

$$\Leftrightarrow \left(\begin{array}{l}\forall\, \mathtt{df} : \mathrm{defs}(\mathtt{x}, \mathtt{P});\ \mathtt{u} : \text{e-uses}(\mathtt{x}, \mathtt{P});\ \mathtt{p} : \text{all-du-sub-path}(\mathtt{x}, \mathtt{P}, \mathtt{df}, \mathtt{u}) \bullet\\ \quad \exists\, \mathtt{spt} : \mathit{SSPT};\ \mathtt{spt_1}, \mathtt{spt_2} : \mathrm{seq}\ \mathit{TLabel} \bullet \mathtt{spt} = \mathit{spt_1} \frown p \frown \mathit{spt_2}\end{array}\right)$$

[definition of all-du-sub-path$(\mathtt{x}, \mathtt{P}, \mathtt{df}, \mathtt{u})$]

Case all-paths ensures all-uses

$$\left(\begin{array}{l}\forall\, p : \text{all-du-sub-path}(\mathtt{x}, \mathtt{P}, \mathtt{df}, \mathtt{u}) \bullet\\ \quad \exists\, \mathtt{spt} : \mathit{SSPT};\ \mathtt{spt_1}, \mathtt{spt_2} : \mathrm{seq}\ \mathit{TLabel} \bullet \mathtt{spt} = \mathit{spt_1} \frown p \frown \mathit{spt_2}\end{array}\right)$$

$$\Leftrightarrow \left(\begin{array}{l}\forall p : \mathrm{seq}\ \mathit{TLabel} \bullet \\ \left(\left(\begin{array}{l}\exists\, \mathtt{spt} : \mathit{sptraces}(\mathtt{P});\ i : 1 \mathbin{..} \#\, \mathit{spt};\ j : (i+1) \mathbin{..} \#\, \mathit{spt} \bullet \\ \quad \mathit{spt}\ i = \mathtt{df} \land \mathit{spt}\ j = \mathtt{u} \land \\ \quad (\forall k : (i+1) \mathbin{..} (j-1) \bullet \mathit{spt}\ k \notin \mathrm{defs}(\mathtt{x}, \mathtt{P})) \land \\ \quad p = (i \mathbin{..} j) \upharpoonright \mathit{spt}\end{array}\right) \Rightarrow \right. \\ \quad \left.\left(\begin{array}{l}\exists\, \mathtt{spt} : \mathit{SSPT};\ \mathtt{spt}_1, \mathtt{spt}_2 : \mathrm{seq}\ \mathit{TLabel} \bullet \\ \quad \mathtt{spt} = \mathit{spt}_1 \frown p \frown \mathit{spt}_2\end{array}\right)\right)\end{array}\right)$$

[definition of all-du-sub-path$(\mathtt{x}, \mathtt{P}, \mathtt{df}, \mathtt{u})$]

$$\Leftrightarrow \left(\begin{array}{l}\forall p : \mathrm{seq}\ \mathit{TLabel} \bullet \\ \left(\left(\begin{array}{l}\exists\, \mathtt{spt} : \mathit{sptraces}(\mathtt{P});\ i : 1 \mathbin{..} \#\, \mathit{spt};\ j : (i+1) \mathbin{..} \#\, \mathit{spt} \bullet \\ \quad \mathit{spt}\ i = \mathtt{df} \land \mathit{spt}\ j = \mathtt{u} \land \\ \quad (\forall k : (i+1) \mathbin{..} (j-1) \bullet \mathit{spt}\ k \notin \mathrm{defs}(\mathtt{x}, \mathtt{P})) \land \\ \quad p = (i \mathbin{..} j) \upharpoonright \mathit{spt}\end{array}\right) \Rightarrow \right. \\ \quad \left.\left(\begin{array}{l}\exists\, \mathtt{spt} : \mathit{SSPT};\ \mathtt{spt}_1, \mathtt{spt}_2 : \mathrm{seq}\ \mathit{TLabel} \bullet \\ \quad \mathtt{spt} = \mathit{spt}_1 \frown p \frown \mathit{spt}_2 \land \text{clear-path}(\mathtt{p}, \mathtt{df}, \mathtt{u}, \mathtt{x})\end{array}\right)\right)\end{array}\right)$$

[clear-path$(\mathtt{spt}, \mathtt{df}, \mathtt{u}, \mathtt{x})$ and $p = (i \mathbin{..} j) \upharpoonright \mathit{spt}$]

$$\Leftrightarrow \left(\begin{array}{l}\forall p : \mathrm{seq}\ \mathit{TLabel};\ \mathtt{spt} : \mathit{sptraces}(\mathtt{P});\ i : 1 \mathbin{..} \#\, \mathit{spt};\ j : (i+1) \mathbin{..} \#\, \mathit{spt} \bullet \\ \left(\left(\begin{array}{l}\mathit{spt}\ i = \mathtt{df} \land \mathit{spt}\ j = \mathtt{u} \land \\ (\forall k : (i+1) \mathbin{..} (j-1) \bullet \mathit{spt}\ k \notin \mathrm{defs}(\mathtt{x}, \mathtt{P})) \land \\ p = (i \mathbin{..} j) \upharpoonright \mathit{spt}\end{array}\right) \Rightarrow \right. \\ \quad \left.\left(\begin{array}{l}\exists\, \mathtt{spt} : \mathit{SSPT};\ \mathtt{spt}_1, \mathtt{spt}_2 : \mathrm{seq}\ \mathit{TLabel} \bullet \\ \quad \mathtt{spt} = \mathit{spt}_1 \frown p \frown \mathit{spt}_2 \land \text{clear-path}(\mathtt{p}, \mathtt{df}, \mathtt{u}, \mathtt{x})\end{array}\right)\right)\end{array}\right)$$

[predicate calculus]

$$\Leftrightarrow \left(\begin{array}{l}\forall\, \mathtt{spt}_3 : \mathit{sptraces}(\mathtt{P});\ i : 1 \mathbin{..} \#\, \mathit{spt}_3;\ j : (i+1) \mathbin{..} \#\, \mathit{spt}_3 \bullet \\ \left(\left(\begin{array}{l}\mathit{spt}_3\ i = \mathtt{df} \land \mathit{spt}_3\ j = \mathtt{u} \land \\ \forall k : (i+1) \mathbin{..} (j-1) \bullet \mathit{spt}_3\ k \notin \mathrm{defs}(\mathtt{x}, \mathtt{P})\end{array}\right) \Rightarrow \right. \\ \quad \left.\left(\begin{array}{l}\exists\, \mathtt{spt} : \mathit{SSPT};\ \mathtt{spt}_1, \mathtt{spt}_2 : \mathrm{seq}\ \mathit{TLabel} \bullet \\ \quad \left(\begin{array}{l}\mathtt{spt} = \mathit{spt}_1 \frown (i \mathbin{..} j) \upharpoonright \mathit{spt} \frown \mathit{spt}_2 \land \\ \text{clear-path}((i \mathbin{..} j) \upharpoonright \mathit{spt}_3, \mathtt{df}, \mathtt{u}, \mathtt{x})\end{array}\right)\end{array}\right)\right)\end{array}\right)$$

[predicate calculus]

$$\Leftrightarrow \left(\begin{array}{l}\forall\, \mathtt{spt}_3 : \mathit{sptraces}(\mathtt{P});\ i : 1 \mathbin{..} \#\, \mathit{spt}_3;\ j : (i+1) \mathbin{..} \#\, \mathit{spt}_3 \bullet \\ \left(\left(\begin{array}{l}\mathit{spt}_3\ i = \mathtt{df} \land \mathit{spt}_3\ j = \mathtt{u} \land \\ \forall k : (i+1) \mathbin{..} (j-1) \bullet \mathit{spt}_3\ k \notin \mathrm{defs}(\mathtt{x}, \mathtt{P})\end{array}\right) \Rightarrow \right. \\ \quad \left.\left(\begin{array}{l}\exists\, \mathtt{spt} : \mathit{SSPT};\ \mathtt{spt}_1, \mathtt{spt}_2 : \mathrm{seq}\ \mathit{TLabel} \bullet \\ \quad \left(\begin{array}{l}\mathtt{spt} = \mathit{spt}_1 \frown (i \mathbin{..} j) \upharpoonright \mathit{spt} \frown \mathit{spt}_2 \land \\ \text{clear-path}(\mathtt{spt}, \mathtt{df}, \mathtt{u}, \mathtt{x})\end{array}\right)\end{array}\right)\right)\end{array}\right)$$

[property of clear-path and $(i \mathbin{..} j) \upharpoonright \mathit{spt}_3$ is a subsequence of $\mathtt{spt}$]

$$\Rightarrow \left(\begin{array}{l}\forall \mathtt{spt_3} : \mathit{sptraces}(\mathtt{P});\ i : 1 \mathrel{..} \#\, \mathit{spt_3};\ j : (i+1) \mathrel{..} \#\, \mathit{spt_3} \bullet \\ \quad \left(\begin{array}{l}\left(\begin{array}{l}\mathit{spt_3}\, i = \mathtt{df} \land \mathit{spt_3}\, j = \mathtt{u} \land \\ \forall k : (i+1) \mathrel{..} (j-1) \bullet \mathit{spt_3}\, k \notin \mathrm{defs}(\mathtt{x}, \mathtt{P})\end{array}\right) \Rightarrow \\ \quad \left(\begin{array}{l}(\exists \mathtt{spt} : \mathit{SSPT} \bullet \text{clear-path}(\mathtt{spt}, \mathtt{df}, \mathtt{u}, \mathtt{x})) \land \\ (\exists \mathtt{spt} : \mathit{sptraces}(\mathtt{P}) \bullet \text{clear-path}(\mathtt{spt}, \mathtt{df}, \mathtt{u}, \mathtt{x}))\end{array}\right)\end{array}\right)\end{array}\right)$$

[predicate calculus]

$$\Leftrightarrow \left(\begin{array}{l}\left(\begin{array}{l}\exists \mathtt{spt_3} : \mathit{sptraces}(\mathtt{P});\ i : 1 \mathrel{..} \#\, \mathit{spt_3};\ j : (i+1) \mathrel{..} \#\, \mathit{spt_3} \bullet \\ \quad \left(\begin{array}{l}\mathit{spt_3}\, i = \mathtt{df} \land \mathit{spt_3}\, j = \mathtt{u} \land \\ \forall k : (i+1) \mathrel{..} (j-1) \bullet \mathit{spt_3}\, k \notin \mathrm{defs}(\mathtt{x}, \mathtt{P})\end{array}\right)\end{array}\right) \Rightarrow \\ \quad \left(\begin{array}{l}(\exists \mathtt{spt} : \mathit{SSPT} \bullet \text{clear-path}(\mathtt{spt}, \mathtt{df}, \mathtt{u}, \mathtt{x})) \land \\ (\exists \mathtt{spt} : \mathit{sptraces}(\mathtt{P}) \bullet \text{clear-path}(\mathtt{spt}, \mathtt{df}, \mathtt{u}, \mathtt{x}))\end{array}\right)\end{array}\right)$$

[predicate calculus]

$$\Leftrightarrow \left(\begin{array}{l}(\exists \mathtt{spt} : \mathit{sptraces}(\mathtt{P}) \bullet \text{clear-path}(\mathtt{spt}, \mathtt{df}, \mathtt{u}, \mathtt{x})) \Rightarrow \\ \quad \left(\begin{array}{l}(\exists \mathtt{spt} : \mathit{SSPT} \bullet \text{clear-path}(\mathtt{spt}, \mathtt{df}, \mathtt{u}, \mathtt{x})) \land \\ (\exists \mathtt{spt} : \mathit{sptraces}(\mathtt{P}) \bullet \text{clear-path}(\mathtt{spt}, \mathtt{df}, \mathtt{u}, \mathtt{x}))\end{array}\right)\end{array}\right)$$

[definition of clear-path($\mathtt{spt}, \mathtt{df}, \mathtt{u}, \mathtt{x}$)]

$$\Leftrightarrow \left(\begin{array}{l}(\exists \mathtt{spt} : \mathit{sptraces}(\mathtt{P}) \bullet \text{clear-path}(\mathtt{spt}, \mathtt{df}, \mathtt{u}, \mathtt{x})) \Rightarrow \\ \quad (\exists \mathtt{spt} : \mathit{SSPT} \bullet \text{clear-path}(\mathtt{spt}, \mathtt{df}, \mathtt{u}, \mathtt{x}))\end{array}\right)$$

[predicate calculus]

Case all-uses ensures all-defs

$$\left(\begin{array}{l}\forall \mathtt{u} : \text{e-uses}(\mathtt{x}, \mathtt{P}) \bullet \\ \quad (\exists \mathtt{spt} : \mathit{sptraces}(\mathtt{P}) \bullet \text{clear-path}(\mathtt{spt}, \mathtt{df}, \mathtt{u}, \mathtt{x})) \Rightarrow \\ \quad\quad (\exists \mathtt{spt} : \mathit{SSPT} \bullet \text{clear-path}(\mathtt{spt}, \mathtt{df}, \mathtt{u}, \mathtt{x}))\end{array}\right)$$

$$\Rightarrow \left(\begin{array}{l}\forall \mathtt{u} : \text{e-uses}(\mathtt{x}, \mathtt{P}) \bullet \\ \quad (\exists \mathtt{spt} : \mathit{sptraces}(\mathtt{P}) \bullet \text{clear-path}(\mathtt{spt}, \mathtt{df}, \mathtt{u}, \mathtt{x})) \Rightarrow \\ \quad\quad (\exists \mathtt{spt} : \mathit{SSPT};\ \mathtt{u} : \text{e-uses}(\mathtt{x}, \mathtt{P}) \bullet \text{clear-path}(\mathtt{spt}, \mathtt{df}, \mathtt{u}, \mathtt{x}))\end{array}\right)$$

[predicate calculus]

$$\Leftrightarrow \left(\begin{array}{l}(\exists \mathtt{spt} : \mathit{sptraces}(\mathtt{P});\ \mathtt{u} : \text{e-uses}(\mathtt{x}, \mathtt{P}) \bullet \text{clear-path}(\mathtt{spt}, \mathtt{df}, \mathtt{u}, \mathtt{x})) \Rightarrow \\ \quad (\exists \mathtt{spt} : \mathit{SSPT};\ \mathtt{u} : \text{e-uses}(\mathtt{x}, \mathtt{P}) \bullet \text{clear-path}(\mathtt{spt}, \mathtt{df}, \mathtt{u}, \mathtt{x}))\end{array}\right)$$

[predicate calculus]

□

4.3 Examples

The very basic var-df-chains, where the same variable is considered as the starting definition and the final use, with a clear path with respect to this variable in between, are covered by the above criteria. More interesting are those var-df-chains where intermediate variables that are defined and then used introduce dependencies between definition and use of different variables.

A first example is the definition of a in $\mathtt{inc?c?a}$ and the use of *notes* in $\mathtt{cash!notes}$. The dependency comes from the fact that *notes* belongs to the set of written variables of *Dispense*, a is an input of this schema, and within $\mathtt{Dispense}$, there is the constraint: $\Sigma\, notes! = a?$. Thus affects($\mathtt{a}, \mathtt{notes}, \mathtt{Dispense}$) holds, and since $\mathtt{cash!notes} \in$ e-uses($\mathtt{notes}, \mathtt{CashMachine}$), we have the var-df-chain:

$$\langle \mathtt{inc?c?a}, \mathtt{var\ notes}, \mathtt{Dispense}, \mathtt{notes} \neq [\![\ ]\!], \mathtt{cash!notes} \rangle$$

We note that its coverage is required by all-du-paths by accident, because it is part of a clear path between a definition of c and a use of c. Here, the effect of the definition of a on the value of *notes* is explicitly required to be covered. The var-df-chains identified below, however, are not required to be covered by the previous criteria. They give rise to new tests.

Other examples of var-df-chains are introduced by the definitions of *nBank*. The label $\mathtt{nBank} := \{ 10 \mapsto \mathtt{cap}, 20 \mapsto \mathtt{cap}, 50 \mapsto \mathtt{cap} \}$ is such a definition, and *nBank* is used in $\mathtt{Dispense}$. Moreover, *notes* is externally used in the label $\mathtt{cash!notes}$. This leads to the following var-df-chain.

$$\langle \mathtt{nBank} := \{ 10 \mapsto \mathtt{cap}, 20 \mapsto \mathtt{cap}, 50 \mapsto \mathtt{cap} \}, \\ \mathtt{inc?c?a}, \mathtt{var\ notes}, \mathtt{Dispense}, \mathtt{notes} \neq [\![\ ]\!], \mathtt{cash!notes} \rangle$$

Coverage of this chain leads to coverage of the effect of a *refill* event, after which the value of *nBank* is updated as indicated above. An initialisation trace that leads to the above var-df-chain is simply $\langle refill \rangle$.

Another definition of the *nBank* variable is in the *DispenseNotes* schema, namely: $nBank'\ n = (nBank\ n) - (notes! \sharp\ n)$. Moreover, *DispenseNotes* also has an internal use of *nBank*. Finally, the path below is clear of *nBank* definitions between the two occurrences of $\mathtt{Dispense}$.

$$\langle \mathtt{Dispense}, \\ \mathtt{notes} \neq [\![\ ]\!], \mathtt{cash!notes}, (\mathtt{outc!c}, \mathtt{tag2}), \mathtt{inc?c?a}, \mathtt{var\ notes}, \mathtt{Dispense} \rangle$$

This leads to the var-df-chain below, where the second occurrence of $\mathtt{Dispense}$ is also taken as a definition of *notes*, which is used externally in the final label.

$$\langle \mathtt{Dispense}, \\ \mathtt{notes} \neq [\![\ ]\!], \mathtt{cash!notes}, (\mathtt{outc!c}, \mathtt{tag2}), \mathtt{inc?c?a}, \mathtt{var\,notes}, \mathtt{Dispense}, \\ \mathtt{notes} \neq [\![\ ]\!], \mathtt{cash!notes} \rangle$$

A possible initialisation trace for this var-df-chain is $\langle \mathtt{inc?c?a} \rangle$.

As already mentioned, our new criterion sel-var-df-chain is inspired by the work in [21], but there are fundamental differences that go beyond the specificities of the *Circus* framework. Because of the nature of *Circus*, it is important not to consider only traces that start with a definition characterised by an input communication like in [21]. The internal state is just as important as any input.

Moreover, throwing away traces that are prefixes of other selected traces like in [21] is not applicable to testing for traces refinement or deadlock reduction (known as the *conf* relation), which are the conformance relations considered

$$\frac{c \wedge (s;\ g)}{(\mathtt{c} \mid \mathtt{s} \models \langle \mathtt{g} \rangle \frown \mathtt{spt}) \xrightarrow{\epsilon}_{ST} (\mathtt{c} \wedge (\mathtt{s};\ \mathtt{g}) \mid \mathtt{s} \models \mathtt{spt})}$$

$$\frac{c \wedge T \neq \varnothing}{(\mathtt{c} \mid \mathtt{s} \models \langle \mathtt{d?x : T} \rangle \frown \mathtt{spt}) \xrightarrow{\mathtt{d?w_0}}_{ST} (\mathtt{c} \wedge \mathtt{w_0} \in \mathtt{T} \mid \mathtt{s};\ \mathtt{var\ x := w_0} \models \mathtt{spt})}$$

$$\frac{c}{(\mathtt{c} \mid \mathtt{s} \models \langle \mathtt{d!e} \rangle \frown \mathtt{spt}) \xrightarrow{\mathtt{d!w_0}}_{ST} (\mathtt{c} \wedge (\mathtt{s};\ \mathtt{w_0} = \mathtt{e}) \mid \mathtt{s} \models \mathtt{spt})}$$

$$\frac{(\mathtt{c_1} \mid \mathtt{s_1} \models \mathtt{A_1}) \xrightarrow{\epsilon} (\mathtt{c_2} \mid \mathtt{s_2} \models \mathtt{Skip})}{(\mathtt{c_1} \mid \mathtt{s_1} \models \langle \mathtt{A_1} \rangle \frown \mathtt{spt}) \xrightarrow{\epsilon}_{ST} (\mathtt{c_2} \mid \mathtt{s_2} \models \mathtt{spt})}$$

Table 2. Operational semantics of sptraces; $\mathtt{w_0}$ stand for fresh symbolic variables

for the *Circus* testing theory [3]. A trace is used to construct tests that check forbidden continuations and required acceptances at a particular point of the SUT history, and that check is not subsumed by tests that arise from longer traces. It is the reason why we do not pursue maximality as in [21].

In the next section, we explain how to obtain cstraces from sptraces (to construct symbolic tests). This is essential for generating tests from the selected sptraces and states the link of these tests with the operational semantics and the *Circus* testing theory, whether data-flow coverage is used for selection or not.

5 Conversion of specification traces to symbolic traces

Converting an sptrace to a symbolic trace requires an operational semantics for sptraces, which we provide in Table 2. It defines a transition relation $\longrightarrow_{ST}$ using four rules: one for when the first label is a guard, two for when it is either an input or an output, and one for an action label $\mathtt{A}$. In this last case, the rules of the operational semantics transition rule $\longrightarrow$ define the new transition relation.

Like in the operational semantics, the configuration is a triple, but here, instead of a process or action, we have an sptrace associated with a constraint $\mathtt{c}$ and a state assignment $\mathtt{s}$. From a configuration $(\mathtt{c} \mid \mathtt{s} \models \langle \mathtt{l} \rangle \frown \mathtt{spt})$ with an sptrace $\langle \mathtt{l} \rangle \frown \mathtt{spt}$, we have a transition to a configuration with $\mathtt{spt}$. The new constraint and state depend on the label $\mathtt{l}$.

For a guard, a transition requires that c is satisfiable and g holds in the current state $(s;\ g)$. In this case, the transition is silent: it has label ϵ.

Input and output communications give rise to non-silent transitions with labels like those of the operational semantics: symbolic inputs and outputs. Inputs $\mathtt{d?x:\ T}$ are annotated with the type T of channel d. The new constraint records that the input value represented by the fresh symbolic variable w_0 has type T and the state is enriched with a declaration of x whose initial value is set to w_0.

$$\frac{(\mathtt{c}_1 \mid \mathtt{s}_1 \models \mathtt{spt}_1) \xrightarrow{\epsilon}_{ST} (\mathtt{c}_2 \mid \mathtt{s}_2 \models \mathtt{spt}_2)}{(\mathtt{c}_1 \mid \mathtt{s}_1 \models \mathtt{spt}_1) \overset{\langle\rangle}{\twoheadrightarrow} (\mathtt{c}_2 \mid \mathtt{s}_2 \models \mathtt{spt}_2)}$$

$$\frac{(\mathtt{c}_1 \mid \mathtt{s}_1 \models \mathtt{spt}_1) \xrightarrow{\mathtt{d}?\alpha_0}_{ST} (\mathtt{c}_2 \mid \mathtt{s}_2 \models \mathtt{spt}_2)}{(\mathtt{c}_1 \mid \mathtt{s}_1 \models \mathtt{spt}_1) \overset{\langle \mathtt{d}?\alpha_0 \rangle}{\twoheadrightarrow} (\mathtt{c}_2 \mid \mathtt{s}_2 \models \mathtt{spt}_2)} \qquad \frac{(\mathtt{c}_1 \mid \mathtt{s}_1 \models \mathtt{spt}_1) \xrightarrow{\mathtt{d}!\alpha_0}_{ST} (\mathtt{c}_2 \mid \mathtt{s}_2 \models \mathtt{spt}_2)}{(\mathtt{c}_1 \mid \mathtt{s}_1 \models \mathtt{spt}_1) \overset{\langle \mathtt{d}.\alpha_0 \rangle}{\twoheadrightarrow} (\mathtt{c}_2 \mid \mathtt{s}_2 \models \mathtt{spt}_2)}$$

$$\frac{(\mathtt{c}_1 \mid \mathtt{s}_1 \models \mathtt{spt}_1) \overset{\mathtt{st}_1}{\twoheadrightarrow} (\mathtt{c}_2 \mid \mathtt{s}_2 \models \mathtt{spt}_2) \quad (\mathtt{c}_2 \mid \mathtt{s}_2 \models \mathtt{spt}_2) \overset{\mathtt{st}_2}{\twoheadrightarrow} (\mathtt{c}_3 \mid \mathtt{s}_3 \models \mathtt{spt}_3)}{(\mathtt{c}_1 \mid \mathtt{s}_1 \models \mathtt{spt}_1) \overset{\mathtt{st}_1 \frown \mathtt{st}_2}{\twoheadrightarrow} (\mathtt{c}_3 \mid \mathtt{s}_3 \models \mathtt{spt}_3)}$$

Table 3. Annotated transition relation: symbolic traces for sptraces

Actions, that is, state operations, are handled like in the operational semantics. We observe that, by definition of the operational semantics [3], all transitions arising from an action in a label are silent and lead to the action **Skip**.

Finally, we have a transition relation $\overset{\mathtt{st}}{\twoheadrightarrow}$ that defines a symbolic trace *st* that captures the interactions corresponding to an sptrace. It is defined in Table 3.

The transition relation $\overset{\mathtt{st}}{\twoheadrightarrow}$ is used below to characterise the cstraces of a process from its sptraces. The function $\mathrm{cstrace_{SPT}}^{\mathtt{a}}(P)$ defines the set of cstraces of P in terms of $\mathit{sptraces}(P)$. The extra parameter $\mathtt{a}$ is an alphabet: a sequence of fresh symbolic variables. The cstraces in $\mathrm{cstrace_{SPT}}^{\mathtt{a}}(P)$ use these variables in the order in which they appear in $\mathtt{a}$.

Definition 19.

$$\begin{array}{l} \mathrm{cstrace_{SPT}}^{\mathtt{a}}(\mathtt{begin\ state}[x : T] \bullet A\ \mathtt{end}) = \\ \quad \mathrm{convSPT}^{\mathtt{a}}(\mathtt{w}_0 \in \mathtt{T}, \mathtt{x} := \mathtt{w}_0)\ (\!|\ \mathit{sptraces}(\mathtt{begin\ state}[x : T] \bullet A\ \mathtt{end})\ |\!) \end{array}$$

As before, we consider a process $\mathtt{begin\ state}[x : T] \bullet A\ \mathtt{end}$ without loss of generality, and define its cstraces by applying a conversion function $\mathrm{convSPT}^{\mathtt{a}}(\mathtt{c}, \mathtt{s})$ to each of its sptraces. It is defined as follows.

Definition 20. *For every alphabet* $\mathtt{a}$*, constraint* $\mathtt{c}$*, state assignment* $\mathtt{s}$ *and sptrace* $\mathtt{spt}$*, we have that* $\mathrm{convSPT}^{\mathtt{a}}(\mathtt{c}, \mathtt{s})\,\mathtt{spt} = (\mathtt{st}, \exists(\alpha\mathtt{c} \setminus \alpha\mathtt{st}) \bullet \mathtt{c}_1)$ *where* $\mathtt{st}$ *and* $\mathtt{c}_1$ *are characterised by* $\alpha\mathtt{st} \le \mathtt{a} \wedge \exists\, \mathtt{s}_1 \bullet (\mathtt{c} \mid \mathtt{s} \models \mathtt{spt}) \overset{\mathtt{st}}{\twoheadrightarrow} (\mathtt{c}_1 \mid \mathtt{s}_1 \models \langle\rangle)$.

Each sptrace gives rise to exactly one cstrace, since any nondeterminism in the actions are captured by the constraint on the symbolic variables. The alphabet $\alpha\mathtt{st}$ of the symbolic trace *st* is a prefix of *a*: $\alpha\mathtt{st} \le \mathtt{a}$.

Example 14. The following cstraces correspond to the sptraces in Example 12.

$$\begin{array}{l} (\langle \mathtt{inc}?\alpha_0?\alpha_1, \mathtt{outc}!\alpha_2 \rangle, \alpha_0 \in \mathtt{CARD} \wedge \alpha_1 \in \mathbb{N}_1 \wedge \alpha_2 = \alpha_0) \\ (\langle \mathtt{inc}?\alpha_0?\alpha_1, \mathtt{cash}!\alpha_2, \mathtt{outc}!\alpha_3 \rangle, \alpha_0 \in \mathtt{CARD} \wedge \alpha_1 \in \mathbb{N}_1 \wedge \\ \;\; \Sigma\alpha_2 = \alpha_1 \wedge (\exists\, \mathtt{w}_0 : \mathtt{Note} \to \mathbb{N} \bullet (\forall\, \mathtt{n} : \mathtt{Note} \bullet \alpha_2 \sharp\, \mathtt{n}) \le \mathtt{w}_0\, \mathtt{n})) \wedge \alpha_3 = \alpha_0) \end{array}$$

We take the alphabet to be $\langle \alpha_0, \alpha_1, \alpha_2, \alpha_3, \ldots \rangle$. The first cstrace comes from both the first and the second sptrace in Example 12. The second cstrace comes from the last sptrace in Example 12. The quantified symbolic variable $\mathtt{w}_0$ represents the internal value of *nBank*, which is not observable in the trace, but contributes to the specification of the observable value α_2.

Two sptraces give rise to the same cstrace because after a withdraw request, the card may be returned immediately for one of two reasons: there is a problem with the card account (like insufficient funds) or there is no money in the cash machine. Since the model abstracts away the existence of accounts and their balances, we cannot distinguish these behaviours by tests from this model.

This is reflected in the fact that the two sptraces have different tags associated with the *outc*!*c* event. This indicates that they correspond to two different parts of the model. This distinction is not testable and that may be a problem for understanding or observing the SUT. A testing tool might, for example, warn that a distinction may need to be introduced or instrumented in the SUT. $\square$

The cstraces defined by the operational semantics capture just observable labels. On the other hand, sptraces were defined specifically to capture the structure of the model, and in doing so, it captures guards and data operations that are not visible in the interface of the SUT. So, it is not surprising that, as illustrated in the above example, there are sptraces that lead to the same cstrace. They correspond to paths in the model that are not distinguishable from the SUT. Requiring their absence in programs is reasonable, but in abstract models that involve nondeterminism, this is not realistic.

The next theorem establishes that tests identified by sptraces are unbiased with respect to refinement, because they specify valid cstraces of the process. Construction of unbiased tests from cstraces was addressed in [3].

Theorem 2 $\mathrm{cstrace}_{\mathrm{SPT}}{}^{\mathtt{a}}(\mathtt{P}) \subseteq \mathit{cstraces}^{\mathtt{a}}(\mathtt{P})$

We do not have equality: there is no empty sptrace, for instance. The main lemma is proved by induction on the specification traces of P.

Proof.

$$\mathrm{cstrace}_{\mathrm{SPT}}{}^{\mathtt{a}}(\mathtt{begin\ state}[\mathtt{x} : \mathtt{T}] \bullet \mathtt{A\ end})$$

$$= \mathrm{convSPT}^{\mathtt{a}}(\mathtt{w}_0 \in \mathtt{T}, \mathtt{x} := \mathtt{w}_0) \,(\!|\, \mathit{sptraces}(\mathtt{begin\ state}[\mathtt{x} : \mathtt{T}] \bullet \mathtt{A\ end}) \,|\!) \qquad [\text{definition of } \mathrm{cstrace}_{\mathrm{SPT}}]$$

$$= \{\ \mathtt{spt} : \mathit{sptraces}(\mathtt{begin\ state}[\mathtt{x} : \mathtt{T}] \bullet \mathtt{A\ end}) \bullet \mathrm{convSPT}^{\mathtt{a}}(\mathtt{w}_0 \in \mathtt{T}, \mathtt{x} := \mathtt{w}_0)\, \mathtt{spt}\ \} \qquad [\text{definition of relational image}]$$

$$
\begin{aligned}
&= \{\ \mathtt{spt} : \mathit{sptraces}(\mathtt{begin\ state}[\mathtt{x} : \mathtt{T}] \bullet \mathtt{A\ end});\ \mathtt{st};\ \mathtt{c}_1;\ \mathtt{s}_1\ | \\
&\qquad\qquad \alpha\mathtt{st} \leq \mathtt{a} \wedge (\mathtt{w}_0 \in \mathtt{T} \mid \mathtt{x} := \mathtt{w}_0 \models \mathtt{spt}) \overset{\mathtt{st}}{\rightarrowtail} (\mathtt{c}_1 \mid \mathtt{s}_1 \models \langle\,\rangle) \\
&\qquad \bullet (\mathtt{st}, \exists(\alpha\mathtt{c} \setminus \alpha\mathtt{st}) \bullet \mathtt{c}_1) \\
&\quad \}
\end{aligned}
$$

[definition of convSPT]

$$
\begin{aligned}
&= \{\ \mathtt{spt};\ \mathtt{st};\ \mathtt{c}_1;\ \mathtt{s}_1;\ \mathtt{c}_2;\ \mathtt{s}_2;\ \mathtt{A}_2\ | \\
&\qquad\qquad (\mathtt{w}_0 \in \mathtt{T} \mid \mathtt{x} := \mathtt{w}_0 \models \mathtt{A}) \overset{\mathtt{spt}}{\twoheadrightarrow} (\mathtt{c}_2 \mid \mathtt{s}_2 \models \mathtt{A}_2) \wedge \mathit{obs}(\mathrm{last}\,\mathtt{spt}) \wedge \\
&\qquad\qquad \alpha\mathtt{st} \leq \mathtt{a} \wedge (\mathtt{w}_0 \in \mathtt{T} \mid \mathtt{x} := \mathtt{w}_0 \models \mathtt{spt}) \overset{\mathtt{st}}{\rightarrowtail} (\mathtt{c}_1 \mid \mathtt{s}_1 \models \langle\,\rangle) \\
&\qquad \bullet (\mathtt{st}, \exists(\alpha\mathtt{c} \setminus \alpha\mathtt{st}) \bullet \mathtt{c}_1) \\
&\quad \}
\end{aligned}
$$

[definition of *sptraces*]

$$
\begin{aligned}
&\subseteq \{\ \mathtt{st};\ \mathtt{c}_1;\ \mathtt{s}_1;\ \mathtt{A}_2\ | \\
&\qquad\qquad \alpha\mathtt{st} \leq \mathtt{a} \wedge (\mathtt{w}_0 \in \mathtt{T} \mid \mathtt{x} := \mathtt{w}_0 \models \mathtt{A}) \overset{\mathtt{st}}{\longrightarrow} (\mathtt{c}_1 \mid \mathtt{s}_1 \models \mathtt{A}_2) \\
&\qquad \bullet (\mathtt{st}, \exists(\alpha\mathtt{c} \setminus \alpha\mathtt{st}) \bullet \mathtt{c}_1) \\
&\quad \}
\end{aligned}
$$

[Lemma 1]

$$
= \mathit{cstraces}^{\mathrm{a}}(\mathtt{begin\ state}[\mathtt{x} : \mathtt{T}] \bullet \mathtt{A\ end}) \qquad \text{[definition of } \mathit{cstraces}]
$$

□

The lemma below establishes the relationship between the transition relations $\twoheadrightarrow$, for specification traces, and $\longrightarrow$, for the operational semantics of actions, which we reproduce in the appendix. The equality between the existential quantifications is semantic equality, not syntactic: the predicates identified by the pieces of text built out of the constraints and traces are equivalent.

Lemma 1.

$$
\begin{aligned}
&\forall\, \mathtt{spt};\ \mathtt{st};\ \mathtt{c}_1;\ \mathtt{s}_1;\ \mathtt{A}_1;\ \mathtt{c}_2;\ \mathtt{s}_2;\ \mathtt{A}_2;\ \mathtt{c}_3;\ \mathtt{s}_3 \bullet \\
&\quad \Big((\mathtt{c}_1 \mid \mathtt{s}_1 \models \mathtt{A}_1) \overset{\mathtt{spt}}{\twoheadrightarrow} (\mathtt{c}_2 \mid \mathtt{s}_2 \models \mathtt{A}_2) \wedge (\mathtt{c}_1 \mid \mathtt{s}_1 \models \mathtt{spt}) \overset{\mathtt{st}}{\rightarrowtail} (\mathtt{c}_3 \mid \mathtt{s}_3 \models \langle\,\rangle) \Big) \Rightarrow \\
&\quad\ \left(\begin{array}{l} \exists\, \mathtt{c}_4, \mathtt{s}_4, \mathtt{A}_4 \bullet \\ \quad (\mathtt{c}_1 \mid \mathtt{s}_1 \models \mathtt{A}_1) \overset{\mathtt{st}}{\longrightarrow} (\mathtt{c}_4 \mid \mathtt{s}_4 \models \mathtt{A}_4) \wedge \\ \quad (\exists(\alpha\mathtt{c}_3 \setminus \alpha\mathtt{st}) \bullet \mathtt{c}_3) = (\exists(\alpha\mathtt{c}_4 \setminus \alpha\mathtt{st}) \bullet \mathtt{c}_4) \end{array} \right)
\end{aligned}
$$

Proof. Direct consequence of Proposition 1 and Lemma 2. □

Lemma 2.

$$
\begin{aligned}
&\forall\, \mathtt{spt};\ \mathtt{st};\ \mathtt{c}_1;\ \mathtt{s}_1;\ \mathtt{A}_1;\ \mathtt{c}_2;\ \mathtt{s}_2;\ \mathtt{A}_2;\ \mathtt{c}_3;\ \mathtt{s}_3 \bullet \\
&\quad ((\mathtt{c}_1 \mid \mathtt{s}_1 \models \mathtt{A}_1) \overset{\mathtt{spt}}{\twoheadrightarrow} (\mathtt{c}_2 \mid \mathtt{s}_2 \models \mathtt{A}_2) \wedge (\mathtt{c}_1 \mid \mathtt{s}_1 \models \mathtt{spt}) \overset{\mathtt{st}}{\rightarrowtail} (\mathtt{c}_3 \mid \mathtt{s}_3 \models \langle\,\rangle)) \Rightarrow \\
&\qquad (\exists(\alpha\mathtt{c}_3 \setminus \alpha\mathtt{st}) \bullet \mathtt{c}_3) = (\exists(\alpha\mathtt{c}_2 \setminus \alpha\mathtt{st}) \bullet \mathtt{c}_2)
\end{aligned}
$$

Proof. By induction on $\mathtt{st}$.

Case $\langle\,\rangle$

$(\mathtt{c}_1 \mid \mathtt{s}_1 \models \mathtt{spt}) \xrightarrow{\langle\,\rangle}\!\!\!\!\rightarrow (\mathtt{c}_3 \mid \mathtt{s}_3 \models \langle\,\rangle)$

In this case, by the definition of $\rightarrow\!\!\!\!\rightarrow$ and $\longrightarrow_{ST}$, $\mathtt{spt}$ is a sequence of guards and actions (without communications). We establish the result by case analysis on $\mathtt{spt}$.

Subcase $\langle \mathrm{g}\rangle \frown \mathtt{spt}$

$(\mathtt{c}_1 \mid \mathtt{s}_1 \models \langle \mathrm{g}\rangle \frown \mathtt{spt}) \xrightarrow{\mathtt{st}}\!\!\!\!\rightarrow (\mathtt{c}_3 \mid \mathtt{s}_3 \models \langle\,\rangle) \wedge (\mathtt{c}_1 \mid \mathtt{s}_1 \models \mathtt{A}_1) \overset{\langle \mathrm{g}\rangle \frown \mathtt{spt}}{\Longrightarrow\!\!\!\!\!\Rightarrow} (\mathtt{c}_2 \mid \mathtt{s}_2 \models \mathtt{A}_2)$

$\Rightarrow$

$$\left(\begin{array}{l} \exists\, \mathtt{A}_3 \bullet \\ \quad \left(\begin{array}{l} (\mathtt{c}_1 \mid \mathtt{s}_1 \models \langle \mathrm{g}\rangle \frown \mathtt{spt}) \xrightarrow{\langle\,\rangle}\!\!\!\!\rightarrow (\mathtt{c}_1 \wedge (\mathtt{s}_1;\ \mathrm{g}) \mid \mathtt{s}_1 \models \mathtt{spt}) \wedge \\ (\mathtt{c}_1 \wedge (\mathtt{s}_1;\ \mathrm{g}) \mid \mathtt{s}_1 \models \mathtt{spt}) \xrightarrow{\langle\,\rangle}\!\!\!\!\rightarrow (\mathtt{c}_3 \mid \mathtt{s}_3 \models \langle\,\rangle) \wedge \\ (\mathtt{c}_1 \mid \mathtt{s}_1 \models \mathtt{A}_1) \overset{\langle \mathrm{g}\rangle}{\Longrightarrow\!\!\!\!\!\Rightarrow} (\mathtt{c}_1 \wedge (\mathtt{s}_1;\ \mathrm{g}) \mid \mathtt{s}_1 \models \mathtt{A}_3) \wedge \\ (\mathtt{c}_1 \wedge (\mathtt{s}_1;\ \mathrm{g}) \mid \mathtt{s}_1 \models \mathtt{A}_3) \overset{\mathtt{spt}}{\Longrightarrow\!\!\!\!\!\Rightarrow} (\mathtt{c}_2 \mid \mathtt{s}_2 \models \mathtt{A}_2) \end{array}\right) \end{array}\right) \quad \text{[Lemma 3]}$$

$\Rightarrow (\exists\, \alpha c_3 \bullet c_3) = (\exists\, \alpha c_2 \bullet c_2)$ [by induction hypothesis]

Subcase $\langle \mathtt{A}\rangle \frown \mathtt{spt}$, where $\mathtt{A}$ is an action of a label: similar.

Case $\langle \mathtt{d}?\alpha_0\rangle$

$(\mathtt{c}_1 \mid \mathtt{s}_1 \models \mathtt{spt}) \xrightarrow{\langle \mathtt{d}?\alpha_0\rangle}\!\!\!\!\rightarrow (\mathtt{c}_3 \mid \mathtt{s}_3 \models \langle\,\rangle)$

In this case, by the definition of $\rightarrow\!\!\!\!\rightarrow$ and $\longrightarrow_{ST}$, $\mathtt{spt}$ is a sequence of guards and actions that ends with an input on the channel d. We again establish the result by case analysis on $\mathtt{spt}$. If it starts with a guard or an action, the proof is similar to that presented above. For $\langle \mathtt{d?x}\rangle$, we have the following.

$(\mathtt{c}_1 \mid \mathtt{s}_1 \models \langle \mathtt{d?x}\rangle) \xrightarrow{\langle \mathtt{d}?\alpha_0\rangle}\!\!\!\!\rightarrow (\mathtt{c}_3 \mid \mathtt{s}_3 \models \langle\,\rangle) \wedge (\mathtt{c}_1 \mid \mathtt{s}_1 \models \mathtt{A}_1) \overset{\langle \mathtt{d?x}\rangle}{\Longrightarrow\!\!\!\!\!\Rightarrow} (\mathtt{c}_2 \mid \mathtt{s}_2 \models \mathtt{A}_2)$

$\Rightarrow\ \mathtt{c}_3 = \mathtt{c}_1 \wedge \alpha_0 \in T\ \wedge$ [Lemma 4]
$\quad \exists\, \mathtt{A}_3 \bullet (\mathtt{c}_1 \mid \mathtt{s}_1 \models \mathtt{A}_1) \overset{\langle \mathtt{d?x}\rangle}{\Longrightarrow\!\!\!\!\!\Rightarrow} (\mathtt{c}_1 \wedge \alpha_0 \in \mathtt{T} \mid \mathtt{s}_1;\ \mathtt{var\,x} := \alpha_0 \models \mathtt{A}_3)$

$\Rightarrow \mathtt{c}_3 = \mathtt{c}_1 \wedge \alpha_0 \in T \wedge \mathtt{c}_2 = \mathtt{c}_1 \wedge \alpha_0 \in T$ [Proposition 2]

$\Rightarrow \mathtt{c}_3 = \mathtt{c}_2$

$\Rightarrow (\exists(\alpha c_3 \setminus \{\alpha_0\}) \bullet c_3) = (\exists(\alpha c_2 \setminus \{\alpha_0\}) \bullet c_2)$ [predicate calculus]

Case $\langle \mathtt{d}!\alpha_0\rangle$ Similar, but relies on Lemma 5.

Case $\langle \mathtt{d}.\alpha_0\rangle \frown \mathtt{st}$ Here, we use $\mathtt{d}.\alpha_0$ to represent an input, an output or even a synchronisation, in which case there is no communicated value α_0.

$$(\mathtt{c}_1 \mid \mathtt{s}_1 \models \mathtt{A}_1) \overset{\mathtt{spt}}{\Rrightarrow} (\mathtt{c}_2 \mid \mathtt{s}_2 \models \mathtt{A}_2) \wedge (\mathtt{c}_1 \mid \mathtt{s}_1 \models \mathtt{spt}) \overset{\langle \mathtt{d}.\alpha_0\rangle \frown \mathtt{st}}{\twoheadrightarrow} (\mathtt{c}_3 \mid \mathtt{s}_3 \models \langle\,\rangle)$$

$\Rightarrow$

$$\left(\begin{array}{l}\exists\, \mathtt{spt}_1, \mathtt{spt}_2, \mathtt{c}_4, \mathtt{s}_4 \bullet \\ \quad\left(\begin{array}{l}(\mathtt{c}_1 \mid \mathtt{s}_1 \models \mathtt{A}_1) \overset{\mathtt{spt}_1 \frown \mathtt{spt}_2}{\Rrightarrow} (\mathtt{c}_2 \mid \mathtt{s}_2 \models \mathtt{A}_2) \wedge \\ (\mathtt{c}_1 \mid \mathtt{s}_1 \models \mathtt{spt}_1 \frown \mathtt{spt}_2) \overset{\langle \mathtt{d}.\alpha_0\rangle}{\twoheadrightarrow} (\mathtt{c}_4 \mid \mathtt{s}_4 \models \mathtt{spt}_2) \wedge \\ (\mathtt{c}_1 \mid \mathtt{s}_1 \models \mathtt{spt}_2) \overset{\mathtt{st}}{\twoheadrightarrow} (\mathtt{c}_3 \mid \mathtt{s}_3 \models \langle\,\rangle)\end{array}\right)\end{array}\right) \text{[definition of } \twoheadrightarrow]$$

$\Rightarrow$

$$\left(\begin{array}{l}\exists\, \mathtt{spt}_1, \mathtt{spt}_2, \mathtt{c}_4, \mathtt{s}_4, \mathtt{A}_4 \bullet \\ \quad\left(\begin{array}{l}(\mathtt{c}_1 \mid \mathtt{s}_1 \models \mathtt{A}_1) \overset{\mathtt{spt}_1 \frown \mathtt{spt}_2}{\Rrightarrow} (\mathtt{c}_2 \mid \mathtt{s}_2 \models \mathtt{A}_2) \wedge \\ (\mathtt{c}_1 \mid \mathtt{s}_1 \models \mathtt{A}_1) \overset{\mathtt{spt}_1}{\Rrightarrow} (\mathtt{c}_4 \mid \mathtt{s}_4 \models \mathtt{A}_4) \wedge \\ (\mathtt{c}_1 \mid \mathtt{s}_1 \models \mathtt{spt}_1 \frown \mathtt{spt}_2) \overset{\langle \mathtt{d}.\alpha_0\rangle}{\twoheadrightarrow} (\mathtt{c}_4 \mid \mathtt{s}_4 \models \mathtt{spt}_2) \wedge \\ (\mathtt{c}_1 \mid \mathtt{s}_1 \models \mathtt{spt}_2) \overset{\mathtt{st}}{\twoheadrightarrow} (\mathtt{c}_3 \mid \mathtt{s}_3 \models \langle\,\rangle)\end{array}\right)\end{array}\right) \text{[Proposition 3]}$$

$\Rightarrow$

$$\left(\begin{array}{l}\exists\, \mathtt{spt}_1, \mathtt{spt}_2, \mathtt{c}_4, \mathtt{s}_4, \mathtt{A}_4, \mathtt{c}_5, \mathtt{s}_5, \mathtt{A}_5 \bullet \\ \quad\left(\begin{array}{l}(\mathtt{c}_1 \mid \mathtt{s}_1 \models \mathtt{A}_1) \overset{\mathtt{spt}_1}{\Rrightarrow} (\mathtt{c}_5 \mid \mathtt{s}_5 \models \mathtt{A}_5) \wedge \\ (\mathtt{c}_5 \mid \mathtt{s}_5 \models \mathtt{A}_5) \overset{\mathtt{spt}_2}{\Rrightarrow} (\mathtt{c}_2 \mid \mathtt{s}_2 \models \mathtt{A}_2) \wedge \\ (\mathtt{c}_1 \mid \mathtt{s}_1 \models \mathtt{A}_1) \overset{\mathtt{spt}_1}{\Rrightarrow} (\mathtt{c}_4 \mid \mathtt{s}_4 \models \mathtt{A}_4) \wedge \\ (\mathtt{c}_1 \mid \mathtt{s}_1 \models \mathtt{spt}_1 \frown \mathtt{spt}_2) \overset{\langle \mathtt{d}.\alpha_0\rangle}{\twoheadrightarrow} (\mathtt{c}_4 \mid \mathtt{s}_4 \models \mathtt{spt}_2) \wedge \\ (\mathtt{c}_1 \mid \mathtt{s}_1 \models \mathtt{spt}_2) \overset{\mathtt{st}}{\twoheadrightarrow} (\mathtt{c}_3 \mid \mathtt{s}_3 \models \langle\,\rangle)\end{array}\right)\end{array}\right) \text{[definition of } \Rrightarrow]$$

$\Rightarrow$

$$\left(\begin{array}{l}\exists\, \mathtt{spt}_1, \mathtt{spt}_2, \mathtt{c}_4, \mathtt{s}_4, \mathtt{A}_4, \mathtt{s}_5, \mathtt{A}_5 \bullet \\ \quad\left(\begin{array}{l}(\mathtt{c}_1 \mid \mathtt{s}_1 \models \mathtt{A}_1) \overset{\mathtt{spt}_1}{\Rrightarrow} (\mathtt{c}_4 \mid \mathtt{s}_5 \models \mathtt{A}_5) \wedge \\ (\mathtt{c}_5 \mid \mathtt{s}_5 \models \mathtt{A}_5) \overset{\mathtt{spt}_2}{\Rrightarrow} (\mathtt{c}_2 \mid \mathtt{s}_2 \models \mathtt{A}_2) \wedge \\ (\mathtt{c}_1 \mid \mathtt{s}_1 \models \mathtt{A}_1) \overset{\mathtt{spt}_1}{\Rrightarrow} (\mathtt{c}_4 \mid \mathtt{s}_4 \models \mathtt{A}_4) \wedge \\ (\mathtt{c}_1 \mid \mathtt{s}_1 \models \mathtt{spt}_1 \frown \mathtt{spt}_2) \overset{\langle \mathtt{d}.\alpha_0\rangle}{\twoheadrightarrow} (\mathtt{c}_4 \mid \mathtt{s}_4 \models \mathtt{spt}_2) \wedge \\ (\mathtt{c}_1 \mid \mathtt{s}_1 \models \mathtt{spt}_2) \overset{\mathtt{st}}{\twoheadrightarrow} (\mathtt{c}_3 \mid \mathtt{s}_3 \models \langle\,\rangle)\end{array}\right)\end{array}\right) \text{[Proposition 2]}$$

$\Rightarrow (\exists(\alpha\mathtt{c}_3 \setminus \alpha\mathtt{st}) \bullet \mathtt{c}_3) = (\exists(\alpha\mathtt{c}_2 \setminus \alpha\mathtt{st}) \bullet \mathtt{c}_2)$ [induction hypothesis]

$\Rightarrow (\exists(\alpha \mathtt{c}_3 \setminus (\{\alpha_0\} \cup \alpha \mathtt{st})) \bullet \mathtt{c}_3) = (\exists(\alpha \mathtt{c}_2 \setminus (\{\alpha_0\} \cup \alpha \mathtt{st})) \bullet \mathtt{c}_2)$

[predicate calculus]

□

The definitions of the various transition systems ensure the property below.

Proposition 1.

$$\begin{array}{l} \forall\, \mathtt{spt};\ \mathtt{st};\ \mathtt{c}_1;\ \mathtt{s}_1;\ \mathtt{A}_1;\ \mathtt{c}_2;\ \mathtt{s}_2;\ \mathtt{A}_2;\ \mathtt{c}_3;\ \mathtt{s}_3 \bullet \\ \quad ((\mathtt{c}_1 \mid \mathtt{s}_1 \models \mathtt{A}_1) \overset{\mathtt{spt}}{\twoheadrightarrow} (\mathtt{c}_2 \mid \mathtt{s}_2 \models \mathtt{A}_2) \wedge (\mathtt{c}_1 \mid \mathtt{s}_1 \models \mathtt{spt}) \xrightarrow{\mathtt{st}} (\mathtt{c}_3 \mid \mathtt{s}_3 \models \langle\,\rangle)) \\ \quad \Rightarrow \\ \quad (\mathtt{c}_1 \mid \mathtt{s}_1 \models \mathtt{A}_1) \xrightarrow{\mathtt{st}} (\mathtt{c}_2 \mid \mathtt{s}_2 \models \mathtt{A}_2) \end{array}$$

Lemma 3.

$$\begin{array}{l} \forall\, \mathtt{c}_1, \mathtt{s}_1, \mathtt{A}_1, \mathtt{g}, \mathtt{spt}, \mathtt{c}_3, \mathtt{s}_3, \mathtt{A}_3 \mid (\mathtt{c}_1 \mid \mathtt{s}_1 \models \mathtt{A}_1) \overset{\langle \mathtt{g} \rangle \frown \mathtt{spt}}{\twoheadrightarrow} (\mathtt{c}_3 \mid \mathtt{s}_3 \models \mathtt{A}_3) \bullet \\ \quad (\exists\, \mathtt{A}_2 \bullet (\mathtt{c}_1 \mid \mathtt{s}_1 \models \mathtt{A}_1) \overset{\langle \mathtt{g} \rangle}{\twoheadrightarrow} (\mathtt{c}_1 \wedge (\mathtt{s}_1;\ \mathtt{g}) \mid \mathtt{s}_1 \models \mathtt{A}_2)) \end{array}$$

Proof. By induction on $\mathtt{A}_1$, considering the cases where the label can be $\langle \mathtt{g} \rangle$.

Case $\mathtt{g} \mathbin{\&} \mathtt{A}$ Direct from Rule (5) in Appendix B and Proposition 3. In Appendix B we present the transition system that defines the labels determined by an action, and is the basis for the definition of $\twoheadrightarrow$. The relation $\Longrightarrow_P$ defined in Appendix B is a partial relation for actions. It is used in conjunction with the operational semantics (included in Appendix A) to define $\Longrightarrow$.

Case $\mathtt{let\ x} \bullet \mathtt{A}_1$ Direct from Rule (10) in Appendix B and the induction hypothesis.

Case $\mathtt{A}_1;\ \mathtt{B}$ Direct from Rule (11) in Appendix B and the induction hypothesis.

Case $(\mathtt{spar\ v} \mid \mathtt{x} \mid \mathtt{y} \mid \mathtt{x}_1 := \mathtt{z}_1 \bullet \mathtt{A}_1) \llbracket\, \mathtt{cs} \,\rrbracket (\mathtt{spar\ v} \mid \mathtt{y} \mid \mathtt{x} \mid \mathtt{x}_2 := \mathtt{z}_2 \bullet \mathtt{A}_2)$ In Rule (13) of Appendix B, we conclude by the induction hypothesis that $\mathtt{c}_3 = \mathtt{c}_1 \wedge (\mathtt{s}_1;\ \mathbf{end}\,\mathtt{v}, \mathtt{y};\ \mathtt{g})$, which can be simplified as follows.

$$\mathtt{c}_3 = \mathtt{c}_1 \wedge (\mathtt{s}_1;\ \mathbf{end}\,\mathtt{v}, \mathtt{y};\ \mathtt{g})$$

$$= \mathtt{c}_3 = \mathtt{c}_1 \wedge (\mathtt{s}_1;\ \mathtt{g})$$

[since $\mathtt{v}, \mathtt{y}$ are not free in $\mathtt{g}$, because names are not reused in actions]

For the state assignment, the induction hypothesis gives $\mathtt{s}_3 = \mathtt{s}_1;\ \mathbf{end}\,\mathtt{v}, \mathtt{y}$. If $\mathtt{s}_1$ is a statement assignment over $\mathtt{v}$, $\mathtt{y}$ and $\mathtt{x}$, then $\mathtt{s}_1;\ \mathbf{end}\,\mathtt{v}, \mathtt{y} \wedge \mathtt{s}_1;\ \mathbf{end}\,\mathtt{x} = \mathtt{s}_1$.

Case $\mathtt{A_1} \setminus \mathtt{cs}$ Direct from Rule (16) in Appendix B and the induction hypothesis. □

Proposition 2.

$$(\mathtt{c_1} \mid \mathtt{s_1} \models \mathtt{A_1}) \overset{\mathtt{spt}}{\Rrightarrow} (\mathtt{c_2} \mid \mathtt{s_2} \models \mathtt{A_2}) \land (\mathtt{c_1} \mid \mathtt{s_1} \models \mathtt{A_1}) \overset{\mathtt{spt}}{\Rrightarrow} (\mathtt{c_3} \mid \mathtt{s_3} \models \mathtt{A_3})$$
$$\Rightarrow$$
$$\mathtt{c_2} = \mathtt{c_3}$$

This proposition follows from the fact that $\Rrightarrow$ identifies a unique path in $\mathtt{A_1}$ via $\mathtt{spt}$ and then follows however many silent moves of the operational semantics are possible. These silent moves are for $\mathtt{Skip};\ A$ and $\sqcap$, which do not change $\mathtt{c_1}$. Finally, for $(\mathtt{spar\ v} \mid \mathtt{x} \mid \mathtt{y} \mid \mathtt{x_1} := \mathtt{z_1} \bullet \mathtt{A_1})\ [\![\ \mathtt{cs}\ ]\!]\ (\mathtt{spar\ v} \mid \mathtt{y} \mid \mathtt{x} \mid \mathtt{x_2} := \mathtt{z_2} \bullet \mathtt{A_2})$, a simple induction would justify that the constraint is maintained.

Lemma 4.

$$\forall\, \mathtt{c_1}, \mathtt{s_1}, \mathtt{A_1}, \mathtt{d}, \mathtt{x}, \mathtt{c_3}, \mathtt{s_3}, \mathtt{A_3} \mid (\mathtt{c_1} \mid \mathtt{s_1} \models \mathtt{A_1}) \overset{\langle \mathtt{d?x} \rangle ^\frown \mathtt{spt}}{\Rrightarrow} (\mathtt{c_3} \mid \mathtt{s_3} \models \mathtt{A_3}) \bullet$$
$$(\exists\, \mathtt{A_2}, \alpha_0 \bullet (\mathtt{c_1} \mid \mathtt{s_1} \models \mathtt{A_1}) \overset{\langle \mathtt{d?x} \rangle}{\Rrightarrow} (\mathtt{c_1} \land \alpha_0 \in \mathtt{T} \mid \mathtt{s_1};\ \mathtt{var\, x} := \alpha_0 \models \mathtt{A_2}))$$

Proof. By case analysis on $\mathtt{A_1}$ like in the proof of Lemma 3.

Case $\mathtt{d?x} : \mathtt{T} \longrightarrow \mathtt{A}$ Direct from Rule (7) in Appendix B and Proposition 3.

Cases $\mathtt{let\ x} \bullet \mathtt{A_1}$, $\mathtt{A_1};\ \mathtt{B}$, and $\mathtt{A_1} \setminus \mathtt{cs}$ are similar to those in the proof of Lemma 3. We observe that, in the case of hiding, if $\mathtt{d}$ is in the set $\mathtt{cs}$ of hidden channels, then the communication $\mathtt{d?x}$ cannot be in the trace. So, we conclude that $\mathtt{d}$ is not in the channel.

Case $(\mathtt{spar\ v} \mid \mathtt{x} \mid \mathtt{y} \mid \mathtt{x_1} := \mathtt{z_1} \bullet \mathtt{A_1})\ [\![\ \mathtt{cs}\ ]\!]\ (\mathtt{spar\ v} \mid \mathtt{y} \mid \mathtt{x} \mid \mathtt{x_2} := \mathtt{z_2} \bullet \mathtt{A_2})$ In Rule (13) of Appendix B, we conclude by the induction hypothesis that $\mathtt{c_3} = \mathtt{c_1} \land \alpha_0 \in \mathtt{T}$, as required, and that $\mathtt{s_3} = \mathtt{s_1};\ \mathtt{end\, v, y};\ \mathtt{var\, a} := \alpha_0$. If $\mathtt{s_1}$ is a statement assignment over variables are $\mathtt{v}, \mathtt{y}$ and $\mathtt{x}$, then

$$\mathtt{s_1};\ \mathtt{end\, v, y};\ \mathtt{var\, a} := \alpha_0 \land \mathtt{s_1};\ \mathtt{end\, z}$$

$$= \mathtt{s_1};\ \mathtt{var\, a} := \alpha_0;\ \mathtt{end\, v, y} \land \mathtt{s_1};\ \mathtt{end\, z}$$

$$= \mathtt{s_1};\ \mathtt{var\, a} := \alpha_0$$

□

Lemma 5.

$$\forall\, \mathtt{c_1}, \mathtt{s_1}, \mathtt{A_1}, \mathtt{d}, \mathtt{e}, \mathtt{c_3}, \mathtt{s_3}, \mathtt{A_3} \mid (\mathtt{c_1} \mid \mathtt{s_1} \models \mathtt{A_1}) \overset{\langle \mathtt{d!e} \rangle ^\frown \mathtt{spt}}{\Rrightarrow} (\mathtt{c_3} \mid \mathtt{s_3} \models \mathtt{A_3}) \bullet$$
$$(\exists\, \mathtt{A_2}, \alpha_0 \bullet (\mathtt{c_1} \mid \mathtt{s_1} \models \mathtt{A_1}) \overset{\langle \mathtt{d!e} \rangle}{\Rrightarrow} (\mathtt{c_1} \land (\mathtt{s_1};\ \alpha_0 = \mathtt{e}) \mid \mathtt{s_1} \models \mathtt{A_2}))$$

Proof. By case analysis on $\mathtt{A_1}$. The interesting cases are as follows.

Case $\mathtt{d!e} \longrightarrow \mathtt{A}$ Direct from Rule (6) in Appendix B and Proposition 3.

Case $(\mathtt{spar\ v \mid x \mid y \mid x_1 := z_1 \bullet A_1}) \llbracket \mathtt{cs} \rrbracket (\mathtt{spar\ v \mid y \mid x \mid x_2 := z_2 \bullet A_2})$ If Rule (13) of Appendix B is applicable, the argument is similar to that in the proof of Lemma 3. If Rule (14) is applicable, we conclude by Lemma 4 $\mathtt{c_3} = \mathtt{c_1} \wedge \alpha_0 \in \mathtt{T}$, and that $\mathtt{c_4} = \mathtt{c_1} \wedge (\mathtt{s_1};\ \mathbf{end}\,\mathtt{v,x};\ \alpha_0 = \mathtt{e})$ by the induction hypothesis. Their conjunction can be simplified as follows.

$$\mathtt{c_1} \wedge \alpha_0 \in \mathtt{T} \wedge \mathtt{c_1} \wedge (\mathtt{s_1};\ \mathbf{end}\,\mathtt{v,x};\ \alpha_0 = \mathtt{e})$$

$$= \mathtt{c_1} \wedge (\mathtt{s_1};\ \mathbf{end}\,\mathtt{v,x};\ \alpha_0 = \mathtt{e})$$

$$[\alpha_0 = \mathtt{e} \Rightarrow \alpha_0 \in \mathtt{T} \text{ since the action is well typed}]$$

$$= \mathtt{c_1} \wedge (\mathtt{s_1};\ \alpha_0 = \mathtt{e})$$

$$[\text{since } \mathtt{v,x} \text{ are not free in } \mathtt{e}\text{, because names are not reused in actions}]$$

Moreover, by Lemma 4, $\mathtt{s_3} = \mathtt{s_1};\ \mathbf{end}\,\mathtt{v,y};\ \mathbf{var}\,\mathtt{a} := \alpha_0$, and by the induction hypothesis, $\mathtt{s_4} = \mathtt{s_1};\ \mathbf{var}\,\mathtt{a} := \alpha_0$. If $\mathtt{s_1}$ is a state assignment over variables $\mathtt{v,y}$ and $\mathtt{x}$, then

$$\exists\,\alpha_0 \bullet (\mathtt{s_1};\ \mathbf{end}\,\mathtt{v,y};\ \mathbf{var}\,\mathtt{a} := \alpha_0;\ \alpha_0 = \mathtt{a}) \Leftrightarrow ((\mathtt{s_1};\ \mathbf{var}\,\mathtt{a} := \alpha_0);\ (\alpha_0 = \mathtt{e}))$$

$$= \exists\,\alpha_0 \bullet \mathtt{true} \Leftrightarrow \mathtt{true}$$

$$= \mathtt{true}$$

This gives us the required result for the constraint. For the state assignment, the result is a direct consequence of the definition of Rule (14). □

The definitions of the various transition systems ensure the property below.

Proposition 3. *For every* $\mathtt{spt_1} \in \mathit{sptraces}(\mathtt{c_1}, \mathtt{s_1}, \mathtt{A_1})$

$$(\mathtt{c_1} \mid \mathtt{s_1} \models \mathtt{spt_1}) \overset{\mathtt{st}}{\twoheadrightarrow} (\mathtt{c_2} \mid \mathtt{s_2} \models \mathtt{spt_2}) \Rightarrow$$
$$(\exists\,\mathtt{A_2} \bullet (\mathtt{c_1} \mid \mathtt{s_1} \models \mathtt{A_1}) \overset{\mathtt{spt1-spt_2}}{\Longrightarrow\!\!\!\!\Rightarrow} (\mathtt{c_2} \mid \mathtt{s_2} \models \mathtt{A_2}))$$

We use $\mathtt{spt_1} - \mathtt{spt_2}$, where $\mathtt{spt_2}$ is a suffix of $\mathtt{spt_1}$ to denote a prefix of $\mathtt{spt_1}$: that containing all its elements before $\mathtt{spt_2}$. This concludes our proof of unbias for every selection strategy based on sptraces.

6 Related works

Data-flow analysis of communicating system has raised interest for quite a while. In one of the first works in this area, Reif and Smolka [19] presented a technique based on the construction, from the considered set of communicating processes, of an special directed acyclic graph called the event spanning graph and provided an approximation of the data flow analysis for the case where the only interferences between processes are message primitives.

Concerning programs, we can cite, among many others, [15], where Naumovitch et al. presented a generalisation to concurrent Java programs of an approach where the accuracy of the data-flow analysis based on a data-flow graph can be improved by supplying additional information, expressed as a finite state automata, to represent the possible communications among threads and feasibility constraints. This technique had been extended in [8] and applied to Ada programs. Since then, numerous specialised techniques and tools have been developed for data flow analysis of multithreaded programs.

More recently, Chugh et al. in [5] have shown how to use race detection to determine when data-flow facts may be killed by the actions of other threads. The approach is not tied to any particular concurrency constructs since various race-detection engines can be used. It makes it possible to improve precision and scalability of the data-flow analysis of a large class of concurrent programs.

Concerning models, in [11], Labbé and Gallois address the issue of slicing communicating symbolic automata specifications, more precisely IOSTS, and thus extend data-flow analysis to this kind of models. The emphasis there is on model reduction. Testing is just mentioned as a perspective.

Data-flow based testing for state-based specification languages has been applied to Lotos by Van der Schoot and Ural in [21], to SDL and Estelle (that is, EFSM) by Ural and others in [22], and extended with control dependencies in [10]. As said in Section 4, our sel-var-df-chain-trace selection criterion is inspired from [21], but different, due to the notion of internal state in *Circus* and to the forms of symbolic tests considered in the *Circus* testing theory. These differences, however, should not prevent its extension to control dependencies, possibly by some slight enrichment of our tagged labels. However, our aim in this paper is more to exemplify via data-flow coverage how to relate coverage of the structure of a *Circus* model to the tests derived from its operational semantics than to multiply examples of possible criteria. Thus this extension is not developed here.

In another context, Tse et al. have adapted data-flow testing to service orchestrations specified in WS-BPEL in [12], and to service choreographies in [13]. From the specifications, they build an XPath Rewriting Graph, which captures the specificities of the underlying process algebras, which is very different from *Circus*, with loose coupling between processes, XML messages, and XPath queries. Data-flow entities (that is, defs, uses, and def-clear sub-paths) are then redefined as Q-DEF, Q-USE, and Q-DU, from where data-flow criteria similar to the conventional ones we present in Section 3 are established.

7 Conclusions

We have presented here a framework for selection of tests from *Circus* models based on data-flow coverage criteria for specification traces, which record sequences of guards, communications and actions defined by a model. To illustrate the use of the definitions, we have formalised some coverage criteria, including a new criterion that takes into account specification traces with internal definitions and uses. Proof of unbias of the selected tests is possible due to formal nature of our setting. We have formalised also the procedure to construct *Circus* cstraces (used to construct tests) from our specification traces. Our formal definitions are, in particular, suited for use with the *Circus* testing tool in [9], which is based on a theorem prover, namely Isabelle/HOL.

Many variants of data-flow coverage criteria can be considered in our framework. For instance, we can consider only inputs as definitions, as in [21]; we can also restrict i-useV to uses within predicates in line with the classical p-uses criterion [18]. In these cases, fewer tests are required. As already said, control dependencies as in [10] could be taken into account.

In addition, the specification traces defined in this paper can be used for other selection criteria, data-flow based and other ones as well, since most features of the models are kept. It is our plan to consider a number of selection criteria for *Circus* tests. Besides data-flow coverage, we have already considered criteria based on cstraces, including synchronisation coverage, a specific criterion for coverage of parallelisms. We plan to explore criteria that consider a variety of *Circus* constructs in an integrated way, to include, for instance, notions of Z schema coverage, case splitting in the pre and postcondition of specification statements, control dependencies and test purposes expressed in *Circus*. We plan also to address in a formal framework the problem of monitoring such tests.

Acknowledgments

We warmly thank Frédéric Voisin for several pertinent comments. We are grateful to the Royal Society and the CNRS for funding our collaboration.

References

1. ISO/IEC 13568:2002. Information technology—Z formal specification notation—syntax, type system and semantics. International Standard.
2. A. L. C. Cavalcanti and M.-C. Gaudel. Specification Coverage for Testing in *Circus*. In *UTP*, volume 6445 of *LNCS*, pages 1 – 45. Springer, 2010.
3. A. L. C. Cavalcanti and M.-C. Gaudel. Testing for Refinement in *Circus*. *Acta Informatica*, 48(2):97 – 147, 2011.
4. A. L. C. Cavalcanti, A. C. A. Sampaio, and J. C. P. Woodcock. A Refinement Strategy for *Circus*. *FACJ*, 15(2 - 3):146 – 181, 2003.
5. R. Chugh et al. Dataflow analysis for concurrent programs using datarace detection. In *ACM SIGPLAN PLDI*, pages 316 – 326. ACM, 2008.
6. L. A. Clarke, A. Podgurski, D. J. Richardson, and S. J. Zeil. A Comparison of Data Flow Path Selection Criteria. In *ICSE*, pages 244 – 251, 1985.
7. E. W. Dijkstra. *A Discipline of Programming.* Prentice-Hall, 1976.
8. M. B. Dwyer et al. Flow analysis for verifying properties of concurrent software systems. *ACM ToSEM*, 13(4):359 – 430, 2004.
9. A. Feliachi, M. C. Gaudel, M. Wenzel, and B. Wolff. The *Circus* Testing Theory Revisited in Isabelle/HOL. In *15th ICFEM*, volume 8144 of *LNCS*, pages 243 – 260. Springer, 2013.
10. H. S. Hong and H. Ural. Dependence testing: Extending data flow testing with control dependence. In *TESTCOM*, pages 23 – 39, 2005.

11. S. Labbé and J.-P. Gallois. Slicing communicating automata specifications: polynomial algorithms for model reduction. *FACJ*, 20(6):563 – 595, 2008.
12. L. Mei, W. K. Chan, and T. H. Tse. Data flow testing of service-oriented workflow applications. In *ICSE*, pages 371 – 380, 2008.
13. L. Mei, W. K. Chan, and T. H. Tse. Data flow testing of service choreography. In *ESEC/FSE*, pages 151 – 160, 2009.
14. C. C. Morgan. *Programming from Specifications*. Prentice-Hall, 2nd edition, 1994.
15. G. Naumovich, G. S. Avrunin, and L. A. Clarke. Data flow analysis for checking properties of concurrent Java programs. In *ICSE*, pages 399 – 410. ACM, 1999.
16. M. V. M. Oliveira. *Formal Derivation of State-Rich Reactive Programs Using Circus*. PhD thesis, University of York, 2006.
17. M. V. M. Oliveira, A. L. C. Cavalcanti, and J. C. P. Woodcock. A UTP Semantics for *Circus*. *FACJ*, 21(1-2):3 – 32, 2009.
18. S. Rapps and E. J. Weyuker. Selecting software test data using data flow information. *IEEE TSE*, 11(4):367 – 375, 1985.
19. J. H. Reif and S. A. Smolka. Data flow analysis of distributed communicating processes. *International Journal of Parallel Programming*, 19(1):1 – 30, 1990.
20. A. W. Roscoe. *Understanding Concurrent Systems*. Texts in Computer Science. Springer, 2011.
21. 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.
22. H. Ural, K. Saleh, and A. W. Williams. Test generation based on control and data dependencies within system specifications in SDL. *Computer Communications*, 23(7):609 – 627, 2000.
23. J. C. P. Woodcock and J. Davies. *Using Z—Specification, Refinement, and Proof*. Prentice-Hall, 1996.

A Operational semantics

$$\frac{}{\begin{pmatrix}\mathbf{begin}\\ \quad \mathbf{state}\ [\,\mathtt{x}:\mathtt{T}\,]\\ \quad \bullet\ \mathtt{A}\\ \mathbf{end}\end{pmatrix} \xrightarrow{\epsilon} \begin{pmatrix}\mathbf{begin}\\ \quad \mathbf{state}\ [\,\mathtt{x}:\mathtt{T}\,] \mid \mathsf{loc}\ (\mathtt{w_0} \in \mathtt{T} \mid \mathtt{x} := \mathtt{w_0})\\ \quad \bullet\ \mathtt{A}\\ \mathbf{end}\end{pmatrix}} \tag{1}$$

$$\frac{(\mathtt{c_1} \mid \mathtt{s_1} \models \mathtt{A_1}) \xrightarrow{l} (\mathtt{c_2} \mid \mathtt{s_2} \models \mathtt{A_2})}{\begin{pmatrix}\mathbf{begin}\\ \quad \mathbf{state}\ [\,\mathtt{x}:\mathtt{T}\,]\\ \quad \mid \mathsf{loc}\ (\mathtt{c_1} \mid \mathtt{s_1})\\ \quad \bullet\ \mathtt{A_1}\\ \mathbf{end}\end{pmatrix} \xrightarrow{l} \begin{pmatrix}\mathbf{begin}\\ \quad \mathbf{state}\ [\,\mathtt{x}:\mathtt{T}\,]\\ \quad \mid \mathsf{loc}\ (\mathtt{c_2} \mid \mathtt{s_2})\\ \quad \bullet\ \mathtt{A_2}\\ \mathbf{end}\end{pmatrix}} \tag{2}$$

$$\frac{\mathtt{c} \wedge (\mathtt{s};\ \mathtt{p}) \wedge (\exists\, \mathtt{v}' \bullet \mathtt{s};\ \mathtt{Q})}{(\mathtt{c} \mid \mathtt{s} \models \mathtt{p} \vdash \mathtt{Q}) \xrightarrow{\epsilon} (\mathtt{c} \wedge (\mathtt{s};\ \mathtt{Q}\,[\mathtt{w_0}/\mathtt{v}']) \mid \mathtt{s};\ \mathtt{v} := \mathtt{w_0} \models \mathtt{Skip})} \quad v' = out\alpha s \tag{3}$$

$$\frac{c \wedge \neg\,(s;\; p)}{(\mathtt{c} \mid \mathtt{s} \models \mathtt{p} \vdash \mathtt{Q}) \xrightarrow{\epsilon} (\mathtt{c} \mid \mathtt{s} \models \mathbf{Chaos})} \tag{4}$$

$$\frac{c}{(\mathtt{c} \mid \mathtt{s} \models \mathbf{Chaos}) \xrightarrow{\epsilon} (\mathtt{c} \mid \mathtt{s} \models \mathbf{Chaos})} \tag{5}$$

$$\frac{c}{(\mathtt{c} \mid \mathtt{s} \models \mathtt{v} := \mathtt{e}) \xrightarrow{\epsilon} (\mathtt{c} \wedge (\mathtt{s};\; \mathtt{w}_0 = \mathtt{e}) \mid \mathtt{s};\; \mathtt{v} := \mathtt{w}_0 \models \mathtt{Skip})} \tag{6}$$

$$\frac{c \wedge (s;\; \mathbf{pre}\, Op)}{(\mathtt{c} \mid \mathtt{s} \models \mathtt{Op}) \xrightarrow{\epsilon} (\mathtt{c} \wedge (\mathtt{s};\; \mathtt{Op}\,[\mathtt{w}_0/\mathtt{v}']) \mid \mathtt{s};\; \mathtt{v} := \mathtt{w}_0 \models \mathtt{Skip})} \quad v' = out\alpha s \tag{7}$$

$$\frac{c \wedge \neg\,(s;\; \mathbf{pre}\, Op)}{(\mathtt{c} \mid \mathtt{s} \models \mathtt{Op}) \xrightarrow{\epsilon} (\mathtt{c} \mid \mathtt{s} \models \mathbf{Chaos})} \tag{8}$$

$$\frac{c}{(\mathtt{c} \mid \mathtt{s} \models \mathtt{d!e} \rightarrow \mathtt{A}) \xrightarrow{\mathtt{d!w_0}} (\mathtt{c} \wedge (\mathtt{s};\; \mathtt{w}_0 = \mathtt{e}) \mid \mathtt{s} \models \mathtt{A})} \tag{9}$$

$$\frac{c \wedge T \neq \varnothing \qquad x \notin \alpha s}{(\mathtt{c} \mid \mathtt{s} \models \mathtt{d?x} : \mathtt{T} \rightarrow \mathtt{A}) \xrightarrow{\mathtt{d?w_0}} (\mathtt{c} \wedge \mathtt{w}_0 \in \mathtt{T} \mid \mathtt{s};\; \mathtt{var}\,\mathtt{x} := \mathtt{w}_0 \models \mathtt{let}\ \mathtt{x} \bullet \mathtt{A})} \tag{10}$$

$$\frac{c \wedge T \neq \varnothing \qquad x \notin \alpha s}{(\mathtt{c} \mid \mathtt{s} \models \mathtt{var}\,\mathtt{x} : \mathtt{T} \bullet \mathtt{A}) \xrightarrow{\epsilon} (\mathtt{c} \wedge \mathtt{w}_0 \in \mathtt{T} \mid \mathtt{s};\; \mathtt{var}\,\mathtt{x} := \mathtt{w}_0 \models \mathtt{let}\ \mathtt{x} \bullet \mathtt{A})} \tag{11}$$

$$\frac{(\mathtt{c}_1 \mid \mathtt{s}_1 \models \mathtt{A}_1) \xrightarrow{\mathtt{l}} (\mathtt{c}_2 \mid \mathtt{s}_2 \models \mathtt{A}_2)}{(\mathtt{c}_1 \mid \mathtt{s}_1 \models \mathtt{let}\ \mathtt{x} \bullet \mathtt{A}_1) \xrightarrow{\mathtt{l}} (\mathtt{c}_2 \mid \mathtt{s}_2 \models \mathtt{let}\ \mathtt{x} \bullet \mathtt{A}_2)} \tag{12}$$

$$\frac{c}{(\mathtt{c} \mid \mathtt{s} \models \mathtt{let}\ \mathtt{x} \bullet \mathtt{Skip}) \xrightarrow{\epsilon} (\mathtt{c} \mid \mathtt{s};\; \mathtt{end}\,\mathtt{x} \models \mathtt{Skip})} \tag{13}$$

$$\frac{(\mathsf{c}_1 \mid \mathsf{s}_1 \models \mathtt{A}_1) \xrightarrow{1} (\mathsf{c}_2 \mid \mathsf{s}_2 \models \mathtt{A}_2)}{(\mathsf{c}_1 \mid \mathsf{s}_1 \models \mathtt{A}_1;\ \mathtt{B}) \xrightarrow{1} (\mathsf{c}_2 \mid \mathsf{s}_2 \models \mathtt{A}_2;\ \mathtt{B})} \tag{14}$$

$$\frac{c}{(\mathsf{c} \mid \mathsf{s} \models \mathtt{Skip};\ \mathtt{A}) \xrightarrow{\epsilon} (\mathsf{c} \mid \mathsf{s} \models \mathtt{A})} \tag{15}$$

$$\frac{c}{(\mathsf{c} \mid \mathsf{s} \models \mathtt{A}_1 \sqcap \mathtt{A}_2) \xrightarrow{\epsilon} (\mathsf{c} \mid \mathsf{s} \models \mathtt{A}_1)} \qquad \frac{c}{(\mathsf{c} \mid \mathsf{s} \models \mathtt{A}_1 \sqcap \mathtt{A}_2) \xrightarrow{\epsilon} (\mathsf{c} \mid \mathsf{s} \models \mathtt{A}_2)} \tag{16}$$

$$\frac{c \wedge (s;\ g)}{(\mathsf{c} \mid \mathsf{s} \models \mathsf{g} \mathbin{\&} \mathtt{A}) \xrightarrow{\epsilon} (\mathsf{c} \wedge (\mathsf{s};\ \mathsf{g}) \mid \mathsf{s} \models \mathtt{A})} \tag{17}$$

$$\frac{c}{(\mathsf{c} \mid \mathsf{s} \models \mathtt{A}_1 \mathbin{\Box} \mathtt{A}_2) \xrightarrow{\epsilon} (\mathsf{c} \mid \mathsf{s} \models (\mathsf{loc}\ \mathsf{c} \mid \mathsf{s} \bullet \mathtt{A}_1) \boxplus (\mathsf{loc}\ \mathsf{c} \mid \mathsf{s} \bullet \mathtt{A}_2))} \tag{18}$$

$$\frac{c_1}{(\mathsf{c} \mid \mathsf{s} \models (\mathsf{loc}\ \mathsf{c}_1 \mid \mathsf{s}_1 \bullet \mathbf{Skip}) \boxplus (\mathsf{loc}\ \mathsf{c}_2 \mid \mathsf{s}_2 \bullet \mathtt{A})) \xrightarrow{\epsilon} (\mathsf{c}_1 \mid \mathsf{s}_1 \models \mathbf{Skip})} \tag{19}$$

$$\frac{c_2}{(\mathsf{c} \mid \mathsf{s} \models (\mathsf{loc}\ \mathsf{c}_1 \mid \mathsf{s}_1 \bullet \mathtt{A}) \boxplus (\mathsf{loc}\ \mathsf{c}_2 \mid \mathsf{s}_2 \bullet \mathbf{Skip})) \xrightarrow{\epsilon} (\mathsf{c}_2 \mid \mathsf{s}_2 \models \mathbf{Skip})} \tag{20}$$

$$\frac{(\mathsf{c}_1 \mid \mathsf{s}_1 \models \mathtt{A}_1) \xrightarrow{\epsilon} (\mathsf{c}_3 \mid \mathsf{s}_3 \models \mathtt{A}_3)}{\left(\begin{array}{l} \mathsf{c} \mid \mathsf{s} \\ \models \\ \left(\begin{array}{l} (\mathsf{loc}\ \mathsf{c}_1 \mid \mathsf{s}_1 \bullet \mathtt{A}_1) \\ \boxplus \\ (\mathsf{loc}\ \mathsf{c}_2 \mid \mathsf{s}_2 \bullet \mathtt{A}_2) \end{array}\right) \end{array}\right) \xrightarrow{\epsilon} \left(\begin{array}{l} \mathsf{c} \mid \mathsf{s} \\ \models \\ \left(\begin{array}{l} (\mathsf{loc}\ \mathsf{c}_3 \mid \mathsf{s}_3 \bullet \mathtt{A}_3) \\ \boxplus \\ (\mathsf{loc}\ \mathsf{c}_2 \mid \mathsf{s}_2 \bullet \mathtt{A}_2) \end{array}\right) \end{array}\right)} \tag{21}$$

$$\frac{(\mathsf{c}_2 \mid \mathsf{s}_2 \models \mathtt{A}_2) \xrightarrow{\epsilon} (\mathsf{c}_3 \mid \mathsf{s}_3 \models \mathtt{A}_3)}{\left(\begin{array}{l} \mathsf{c} \mid \mathsf{s} \\ \models \\ \left(\begin{array}{l} (\mathsf{loc}\ \mathsf{c}_1 \mid \mathsf{s}_1 \bullet \mathtt{A}_1) \\ \boxplus \\ (\mathsf{loc}\ \mathsf{c}_2 \mid \mathsf{s}_2 \bullet \mathtt{A}_2) \end{array}\right) \end{array}\right) \xrightarrow{\epsilon} \left(\begin{array}{l} \mathsf{c} \mid \mathsf{s} \\ \models \\ \left(\begin{array}{l} (\mathsf{loc}\ \mathsf{c}_1 \mid \mathsf{s}_1 \bullet \mathtt{A}_1) \\ \boxplus \\ (\mathsf{loc}\ \mathsf{c}_3 \mid \mathsf{s}_3 \bullet \mathtt{A}_3) \end{array}\right) \end{array}\right)} \tag{22}$$

$$\frac{(\mathtt{c}_1 \mid \mathtt{s}_1 \models \mathtt{A}_1) \xrightarrow{\mathtt{l}} (\mathtt{c}_3 \mid \mathtt{s}_3 \models \mathtt{A}_3) \quad \mathtt{l} \neq \epsilon}{(\mathtt{c} \mid \mathtt{s} \models (\mathsf{loc}\ \mathtt{c}_1 \mid \mathtt{s}_1 \bullet \mathtt{A}_1) \boxplus (\mathsf{loc}\ \mathtt{c}_2 \mid \mathtt{s}_2 \bullet \mathtt{A}_2)) \xrightarrow{\mathtt{l}} (\mathtt{c}_3 \mid \mathtt{s}_3 \models \mathtt{A}_3)} \tag{23}$$

$$\frac{(\mathtt{c}_2 \mid \mathtt{s}_2 \models \mathtt{A}_2) \xrightarrow{\mathtt{l}} (\mathtt{c}_3 \mid \mathtt{s}_3 \models \mathtt{A}_3) \quad \mathtt{l} \neq \epsilon}{(\mathtt{c} \mid \mathtt{s} \models (\mathsf{loc}\ \mathtt{c}_1 \mid \mathtt{s}_1 \bullet \mathtt{A}_1) \boxplus (\mathsf{loc}\ \mathtt{c}_2 \mid \mathtt{s}_2 \bullet \mathtt{A}_2)) \xrightarrow{\mathtt{l}} (\mathtt{c}_3 \mid \mathtt{s}_3 \models \mathtt{A}_3)} \tag{24}$$

$$\frac{c}{(\mathtt{c} \mid \mathtt{s} \models \mathtt{A}_1 [\![\mathtt{x}_1 \mid \mathtt{cs} \mid \mathtt{x}_2]\!] \mathtt{A}_2) \xrightarrow{\epsilon} \begin{pmatrix} \mathtt{c} \mid \mathtt{s} \\ \models \\ \quad (\mathtt{par\ s} \mid \mathtt{x}_1 \bullet \mathtt{A}_1) [\![\mathtt{cs}]\!] (\mathtt{par\ s} \mid \mathtt{x}_2 \bullet \mathtt{A}_2) \end{pmatrix}} \tag{25}$$

$$\frac{c}{\begin{pmatrix} \mathtt{c} \mid \mathtt{s} \\ \models \\ \begin{pmatrix} (\mathtt{par\ s}_1 \mid \mathtt{x}_1 \bullet \mathtt{Skip}) \\ [\![\mathtt{cs}]\!] \\ (\mathtt{par\ s}_2 \mid \mathtt{x}_2 \bullet \mathtt{Skip}) \end{pmatrix} \end{pmatrix} \xrightarrow{\epsilon} (\mathtt{c} \mid (\exists\, \mathtt{x}_2' \bullet \mathtt{s}_1) \wedge (\exists\, \mathtt{x}_1' \bullet \mathtt{s}_2) \models \mathtt{Skip})} \tag{26}$$

$$\frac{(\mathtt{c} \mid \mathtt{s}_1 \models \mathtt{A}_1) \xrightarrow{\mathtt{l}} (\mathtt{c}_3 \mid \mathtt{s}_3 \models \mathtt{A}_3) \quad \mathtt{l} = \epsilon \vee \mathrm{chan}\, \mathtt{l} \notin cs}{\begin{pmatrix} \mathtt{c} \mid \mathtt{s} \\ \models \\ \begin{pmatrix} (\mathtt{par\ s}_1 \mid \mathtt{x}_1 \bullet \mathtt{A}_1) \\ [\![\mathtt{cs}]\!] \\ (\mathtt{par\ s}_2 \mid \mathtt{x}_2 \bullet \mathtt{A}_2) \end{pmatrix} \end{pmatrix} \xrightarrow{\mathtt{l}} \begin{pmatrix} \mathtt{c}_3 \mid \mathtt{s} \\ \models \\ \begin{pmatrix} (\mathtt{par\ s}_3 \mid \mathtt{x}_1 \bullet \mathtt{A}_3) \\ [\![\mathtt{cs}]\!] \\ (\mathtt{par\ s}_2 \mid \mathtt{x}_2 \bullet \mathtt{A}_2) \end{pmatrix} \end{pmatrix}} \tag{27}$$

$$\frac{(\mathtt{c} \mid \mathtt{s}_2 \models \mathtt{A}_2) \xrightarrow{\mathtt{l}} (\mathtt{c}_3 \mid \mathtt{s}_3 \models \mathtt{A}_3) \quad \mathtt{l} = \epsilon \vee \mathrm{chan}\, \mathtt{l} \notin cs}{\begin{pmatrix} \mathtt{c} \mid \mathtt{s} \\ \models \\ \begin{pmatrix} (\mathtt{par\ s}_1 \mid \mathtt{x}_1 \bullet \mathtt{A}_1) \\ [\![\mathtt{cs}]\!] \\ (\mathtt{par\ s}_2 \mid \mathtt{x}_2 \bullet \mathtt{A}_2) \end{pmatrix} \end{pmatrix} \xrightarrow{\mathtt{l}} \begin{pmatrix} \mathtt{c}_3 \mid \mathtt{s} \\ \models \\ \begin{pmatrix} (\mathtt{par\ s}_1 \mid \mathtt{x}_1 \bullet \mathtt{A}_1) \\ [\![\mathtt{cs}]\!] \\ (\mathtt{par\ s}_3 \mid \mathtt{x}_2 \bullet \mathtt{A}_3) \end{pmatrix} \end{pmatrix}} \tag{28}$$

$$\frac{\begin{array}{c}\left(\begin{array}{c}(\mathtt{c} \mid \mathtt{s_1} \models \mathtt{A_1}) \xrightarrow{\mathtt{d?w_1}} (\mathtt{c_3} \mid \mathtt{s_3} \models \mathtt{A_3}) \;\wedge\; (\mathtt{c} \mid \mathtt{s_2} \models \mathtt{A_2}) \xrightarrow{\mathtt{d!w_2}} (\mathtt{c_4} \mid \mathtt{s_4} \models \mathtt{A_4}) \\ \vee \\ (\mathtt{c} \mid \mathtt{s_1} \models \mathtt{A_1}) \xrightarrow{\mathtt{d!w_1}} (\mathtt{c_3} \mid \mathtt{s_3} \models \mathtt{A_3}) \;\wedge\; (\mathtt{c} \mid \mathtt{s_2} \models \mathtt{A_2}) \xrightarrow{\mathtt{d?w_2}} (\mathtt{c_4} \mid \mathtt{s_4} \models \mathtt{A_4}) \\ \vee \\ (\mathtt{c} \mid \mathtt{s_1} \models \mathtt{A_1}) \xrightarrow{\mathtt{d!w_1}} (\mathtt{c_3} \mid \mathtt{s_3} \models \mathtt{A_3}) \;\wedge\; (\mathtt{c} \mid \mathtt{s_2} \models \mathtt{A_2}) \xrightarrow{\mathtt{d!w_2}} (\mathtt{c_4} \mid \mathtt{s_4} \models \mathtt{A_4})\end{array}\right) \\ \mathtt{d} \in cs \quad c_3 \wedge c_4 \wedge w_1 = w_2\end{array}}{\left(\begin{array}{l}\mathtt{c} \mid \mathtt{s} \\ \models \\ \left(\begin{array}{c}(\mathtt{par}\ \mathtt{s_1} \mid \mathtt{x_1} \bullet \mathtt{A_1}) \\ [\![\mathtt{cs}]\!] \\ (\mathtt{par}\ \mathtt{s_2} \mid \mathtt{x_2} \bullet \mathtt{A_2})\end{array}\right)\end{array}\right) \xrightarrow{\mathtt{d!w_2}} \left(\begin{array}{l}\mathtt{c_3} \wedge \mathtt{c_4} \wedge \mathtt{w_1} = \mathtt{w_2} \mid \mathtt{s} \\ \models \\ \left(\begin{array}{c}(\mathtt{par}\ \mathtt{s_3} \mid \mathtt{x_1} \bullet \mathtt{A_3}) \\ [\![\mathtt{cs}]\!] \\ (\mathtt{par}\ \mathtt{s_4} \mid \mathtt{x_2} \bullet \mathtt{A_4})\end{array}\right)\end{array}\right)} \tag{29}$$

$$\frac{\begin{array}{c}(\mathtt{c} \mid \mathtt{s_1} \models \mathtt{A_1}) \xrightarrow{\mathtt{d?w_1}} (\mathtt{c_3} \mid \mathtt{s_3} \models \mathtt{A_3}) \quad (\mathtt{c} \mid \mathtt{s_2} \models \mathtt{A_2}) \xrightarrow{\mathtt{d?w_2}} (\mathtt{c_4} \mid \mathtt{s_4} \models \mathtt{A_4}) \\ \mathtt{d} \in cs \quad c_3 \wedge c_4 \wedge w_1 = w_2\end{array}}{\left(\begin{array}{l}\mathtt{c} \mid \mathtt{s} \\ \models \\ \left(\begin{array}{c}(\mathtt{par}\ \mathtt{s_1} \mid \mathtt{x_1} \bullet \mathtt{A_1}) \\ [\![\mathtt{cs}]\!] \\ (\mathtt{par}\ \mathtt{s_2} \mid \mathtt{x_2} \bullet \mathtt{A_2})\end{array}\right)\end{array}\right) \xrightarrow{\mathtt{d?w_2}} \left(\begin{array}{l}\mathtt{c_3} \wedge \mathtt{c_4} \wedge \mathtt{w_1} = \mathtt{w_2} \mid \mathtt{s} \\ \models \\ \left(\begin{array}{c}(\mathtt{par}\ \mathtt{s_3} \mid \mathtt{x_1} \bullet \mathtt{A_3}) \\ [\![\mathtt{cs}]\!] \\ (\mathtt{par}\ \mathtt{s_4} \mid \mathtt{x_2} \bullet \mathtt{A_4})\end{array}\right)\end{array}\right)} \tag{30}$$

$$\frac{(\mathtt{c_1} \mid \mathtt{s_1} \models \mathtt{A_1}) \xrightarrow{\mathtt{l}} (\mathtt{c_2} \mid \mathtt{s_2} \models \mathtt{A_2}) \quad \mathtt{l} \neq \epsilon \quad \text{chan}\ \mathtt{l} \notin cs}{(\mathtt{c_1} \mid \mathtt{s_1} \models \mathtt{A_1} \setminus \mathtt{cs}) \xrightarrow{\mathtt{l}} (\mathtt{c_2} \mid \mathtt{s_2} \models \mathtt{A_2} \setminus \mathtt{cs})} \tag{31}$$

$$\frac{(\mathtt{c_1} \mid \mathtt{s_1} \models \mathtt{A_1}) \xrightarrow{\mathtt{l}} (\mathtt{c_2} \mid \mathtt{s_2} \models \mathtt{A_2}) \quad \mathtt{l} = \epsilon \vee \text{chan}\ \mathtt{l} \in cs}{(\mathtt{c_1} \mid \mathtt{s_1} \models \mathtt{A_1} \setminus \mathtt{cs}) \xrightarrow{\epsilon} (\mathtt{c_2} \mid \mathtt{s_2} \models \mathtt{A_2} \setminus \mathtt{cs})} \tag{32}$$

$$\frac{c}{(\mathtt{c} \mid \mathtt{s} \models \mathtt{Skip} \setminus \mathtt{cs}) \xrightarrow{\epsilon} (\mathtt{c} \mid \mathtt{s} \models \mathtt{Skip})} \tag{33}$$

$$\frac{c}{(\mathtt{c} \mid \mathtt{s} \models \mu\ \mathtt{X} \bullet \mathtt{A}, \delta) \xrightarrow{\epsilon} (\mathtt{c} \mid \mathtt{s} \models \mathtt{A}, \delta \oplus \{\mathtt{X} \mapsto \mathtt{A}\})} \tag{34}$$

$$\frac{c}{(\mathtt{c} \mid \mathtt{s} \models \mathtt{X}, \delta) \xrightarrow{\epsilon} (\mathtt{c} \mid \mathtt{s} \models \delta\ \mathtt{X}, \delta)} \qquad (35)$$

B Specification-oriented transition system: labels

The rule for processes is not specifically for the transition system for the relation $\Longrightarrow_P$ that defines the labels arising from an action. The transition system of $\Longrightarrow_P$ is used in combination with the operational semantics to define maximal non-silent transitions with specification labels. These are characterised by the relation $\Longrightarrow$, for which we define a transition rule for processes.

$$\frac{(\mathit{state}(\mathtt{P}_1) \models \mathit{maction}(\mathtt{P}_1)) \overset{1}{\Longrightarrow} (\mathit{state}(\mathtt{P}_2) \models \mathit{maction}(\mathtt{P}_2))}{\mathtt{P}_1 \overset{1}{\Longrightarrow} \mathtt{P}_2} \qquad (1)$$

It is the transition system for $\Longrightarrow_P$ that is referenced in proofs presented in this report, and we reproduce it below.

$$\frac{c \wedge (s;\ p) \wedge (\exists\, v' \bullet s;\ Q)}{(\mathtt{c} \mid \mathtt{s} \models \mathtt{p} \vdash \mathtt{Q}) \overset{\mathtt{p} \vdash \mathtt{Q}}{\Longrightarrow}_P (\mathtt{c} \wedge (\mathtt{s};\ \mathtt{Q}\,[\mathtt{w}_0/\mathtt{v}'])) \mid \mathtt{s};\ \mathtt{v} := \mathtt{w}_0 \models \mathtt{Skip})} \quad v' = out\alpha s \qquad (2)$$

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

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

$$\frac{c \wedge (s;\ g)}{(\mathtt{c} \mid \mathtt{s} \models \mathtt{g} \mathbin{\&} \mathtt{A}) \overset{\mathtt{g}}{\Longrightarrow}_P (\mathtt{c} \wedge (\mathtt{s};\ \mathtt{g}) \mid \mathtt{s} \models \mathtt{A})} \qquad (5)$$

$$\frac{c}{(\mathtt{c} \mid \mathtt{s} \models \mathtt{d!e} \rightarrow \mathtt{A}) \overset{\mathtt{d!e}}{\Longrightarrow}_P (\mathtt{c} \wedge (\mathtt{s};\ \mathtt{w}_0 = \mathtt{e}) \mid \mathtt{s} \models \mathtt{A})} \qquad (6)$$

$$\frac{c \wedge T \neq \varnothing \qquad x \notin \alpha s}{(\mathtt{c} \mid \mathtt{s} \models \mathtt{d?x} : \mathtt{T} \rightarrow \mathtt{A}) \overset{\mathtt{d?x}}{\Longrightarrow} (\mathtt{c} \wedge \mathtt{w}_0 \in \mathtt{T} \mid \mathtt{s};\ \mathbf{var}\,\mathtt{x} := \mathtt{w}_0 \models \mathtt{let}\ \mathtt{x} \bullet \mathtt{A})} \qquad (7)$$

$$\frac{c \wedge T \neq \varnothing \qquad x \notin \alpha s}{(\mathtt{c} \mid \mathtt{s} \models \mathtt{var\,x : T \bullet A}) \overset{(\mathtt{var\,x:T})}{\Longrightarrow}_P (\mathtt{c} \wedge \mathtt{w_0} \in \mathtt{T} \mid \mathtt{s};\ \mathtt{var\,x := w_0} \models \mathtt{let\ x \bullet A})} \tag{8}$$

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

$$\frac{(\mathtt{c_1} \mid \mathtt{s_1} \models \mathtt{A_1}) \overset{\mathtt{l}}{\Longrightarrow}_P (\mathtt{c_2} \mid \mathtt{s_2} \models \mathtt{A_2})}{(\mathtt{c_1} \mid \mathtt{s_1} \models \mathtt{let\ x \bullet A_1}) \overset{\mathtt{l}}{\Longrightarrow}_P (\mathtt{c_2} \mid \mathtt{s_2} \models \mathtt{let\ x \bullet A_2})} \tag{10}$$

$$\frac{(\mathtt{c_1} \mid \mathtt{s_1} \models \mathtt{A_1}) \overset{\mathtt{l}}{\Longrightarrow}_P (\mathtt{c_2} \mid \mathtt{s_2} \models \mathtt{A_2})}{(\mathtt{c_1} \mid \mathtt{s_1} \models \mathtt{A_1};\ \mathtt{B}) \overset{\mathtt{l}}{\Longrightarrow}_P (\mathtt{c_2} \mid \mathtt{s_2} \models \mathtt{A_2};\ \mathtt{B})} \tag{11}$$

$$\frac{c}{\begin{array}{l} (\mathtt{c} \mid \mathtt{s} \models \mathtt{A_1} \llbracket \mathtt{x1} \mid \mathtt{cs} \mid \mathtt{x2} \rrbracket \mathtt{A_2}) \\ \overset{\mathtt{var}\ \mathtt{v_l,v_r := v,v}}{\Longrightarrow}_P \\ \left(\begin{array}{l} \mathtt{c} \mid \mathtt{s};\ \mathtt{var\,v_l, v_r := v, v} \\ \models \\ \left(\begin{array}{c} (\mathtt{spar\ v} \mid \mathtt{v_l} \mid \mathtt{v_r} \mid \mathtt{x1 := x1_l} \bullet \mathtt{A_1[v_l/v]}) \\ \llbracket \mathtt{cs} \rrbracket \\ (\mathtt{spar\ v} \mid \mathtt{v_r} \mid \mathtt{v_l} \mid \mathtt{x2 := x2_r} \bullet \mathtt{A_2[v_r/v]}) \end{array} \right) \end{array} \right) \end{array}} \quad \begin{array}{l} v' = out\alpha s \\ v = x1, x2 \\ \mathit{fresh}\ v_l, v_r \end{array} \tag{12}$$

$$\frac{(\mathtt{c} \mid \mathtt{s};\ \mathtt{end\,v, y} \models \mathtt{A_1}) \overset{\mathtt{l}}{\Longrightarrow}_P (\mathtt{c_3} \mid \mathtt{s_3} \models \mathtt{A_3}) \quad \mathrm{chan}\,\mathtt{l} = \epsilon \vee \mathrm{chan}\,\mathtt{l} \notin cs}{\begin{array}{l} (\mathtt{c} \mid \mathtt{s} \models (\mathtt{spar\ v} \mid \mathtt{x} \mid \mathtt{y} \mid \mathtt{x_1 := z_1} \bullet \mathtt{A_1}) \llbracket \mathtt{cs} \rrbracket (\mathtt{spar\ v} \mid \mathtt{y} \mid \mathtt{x} \mid \mathtt{x_2 := z_2} \bullet \mathtt{A_2})) \\ \overset{\mathtt{l}}{\Longrightarrow}_P \\ \left(\begin{array}{l} \mathtt{c_3} \mid \mathtt{s_3} \wedge \mathtt{s};\ \mathtt{end\,x} \\ \models \\ \left(\begin{array}{c} (\mathtt{spar\ v} \mid \mathtt{x} \restriction (end\,\mathtt{l}), (var\,\mathtt{l}) \mid \mathtt{y} \mid \mathtt{x_1 := z_1} \bullet \mathtt{A_3}) \\ \llbracket \mathtt{cs} \rrbracket \\ (\mathtt{spar\ v} \mid \mathtt{y} \mid \mathtt{x} \restriction (end\,\mathtt{l}), (var\,\mathtt{l}) \mid \mathtt{x_2 := z_2} \bullet \mathtt{A_2}) \end{array} \right) \end{array} \right) \end{array}} \tag{13}$$

$$\frac{\begin{array}{c}(\mathtt{c} \mid \mathtt{s};\ \mathbf{end}\,\mathtt{v},\mathtt{y} \models \mathtt{A}_1) \xRightarrow{\mathtt{d?a}}_P (\mathtt{c}_3 \mid \mathtt{s}_3 \models \mathtt{A}_3)\\ (\mathtt{c} \mid \mathtt{s};\ \mathbf{end}\,\mathtt{v},\mathtt{x} \models \mathtt{A}_2) \xRightarrow{\mathtt{d!e}}_P (\mathtt{c}_4 \mid \mathtt{s}_4 \models \mathtt{A}_4)\\ \mathtt{d} \in cs \quad c_3 \wedge c_4 \wedge \exists\, w_0 \bullet (s_3;\ (w_0 = a)) \Leftrightarrow (s_4;\ (w_0 = e))\end{array}}{\begin{array}{l}(\mathtt{c} \mid \mathtt{s} \models (\mathbf{spar}\ \mathtt{v} \mid \mathtt{x} \mid \mathtt{y} \mid \mathtt{x}_1 := \mathtt{z}_1 \bullet \mathtt{A}_1) \llbracket\, \mathtt{cs} \,\rrbracket (\mathbf{spar}\ \mathtt{v} \mid \mathtt{y} \mid \mathtt{x} \mid \mathtt{x}_2 := \mathtt{z}_2 \bullet \mathtt{A}_2))\\ \xRightarrow{\mathtt{d!e}}_P\\ \quad\left(\begin{array}{l}\mathtt{c}_3 \wedge \mathtt{c}_4 \wedge \exists\, \mathtt{w}_0 \bullet (\mathtt{s}_3;\ (\mathtt{w}_0 = \mathtt{a})) \Leftrightarrow (\mathtt{s}_4;\ (\mathtt{w}_0 = \mathtt{e})) \mid \mathtt{s}\\ \models\\ \left(\begin{array}{c}(\mathbf{spar}\ \mathtt{v} \mid \mathtt{x} \upharpoonright \mathtt{a} \mid \mathtt{y} \mid \mathtt{x}_1 := \mathtt{z}_1 \bullet \mathtt{A}_3)\\ \llbracket \mathtt{cs} \mid \mathbf{var}\,\mathtt{a} := \mathtt{e} \mid \mathtt{s}_3 \wedge \mathtt{s}_4 \wedge \mathtt{s};\ \mathbf{end}\,\mathtt{x},\mathtt{y} \rrbracket\\ (\mathbf{spar}\ \mathtt{v} \mid \mathtt{y} \mid \mathtt{x} \upharpoonright \mathtt{a} \mid \mathtt{x}_2 := \mathtt{z}_1 \bullet \mathtt{A}_4)\end{array}\right)\end{array}\right)\end{array}} \tag{14}$$

The above rule uses a new parallel construct that keeps track of the new input variable declared and the new state obtained as a consequence. It is used to ensure that, as required here, all transitions have a single label, and the label contains a guard, a communication, or an action. The next rule ensures that in the next step of the evaluation of the parallelism, the variable declaration and state change are recorded. This concern was not present in [2].

$$\frac{c}{\begin{array}{l}(\mathtt{c} \mid \mathtt{s} \models \left(\begin{array}{c}(\mathbf{spar}\ \mathtt{v} \mid \mathtt{x} \mid \mathtt{y} \mid \mathtt{x}_1 := \mathtt{z}_1 \bullet \mathtt{A}_1)\\ \llbracket \mathtt{cs} \mid \mathbf{var}\,\mathtt{a} := \mathtt{e} \mid \mathtt{s}_1 \rrbracket\\ (\mathbf{spar}\ \mathtt{v} \mid \mathtt{y} \mid \mathtt{x} \mid \mathtt{x}_2 := \mathtt{z}_2 \bullet \mathtt{A}_2)\end{array}\right))\\ \xRightarrow{\mathbf{var}\ \mathtt{a} := \mathtt{e}}_P\\ \quad (\mathtt{c} \mid \mathtt{s}_1 \models (\mathbf{spar}\ \mathtt{v} \mid \mathtt{x} \mid \mathtt{y} \mid \mathtt{x}_1 := \mathtt{z}_1 \bullet \mathtt{A}_1) \llbracket\, \mathtt{cs} \,\rrbracket (\mathbf{spar}\ \mathtt{v} \mid \mathtt{y} \mid \mathtt{x} \mid \mathtt{x}_2 := \mathtt{z}_2 \bullet \mathtt{A}_2))\end{array}} \tag{15}$$

Rules similar to those above for parallelism are needed for external choice.

$$\frac{(\mathtt{c}_1 \mid \mathtt{s}_1 \models \mathtt{A}_1) \xRightarrow{\mathtt{l}}_P (\mathtt{c}_2 \mid \mathtt{s}_2 \models \mathtt{A}_2) \quad \mathrm{chan}\ \mathtt{l} \notin cs}{(\mathtt{c}_1 \mid \mathtt{s}_1 \models \mathtt{A}_1 \setminus \mathtt{cs}) \xRightarrow{\mathtt{l}}_P (\mathtt{c}_2 \mid \mathtt{s}_2 \models \mathtt{A}_2 \setminus \mathtt{cs})} \tag{16}$$

Above, we assume that, if $\mathtt{l}$ is not a communication, then chan $\mathtt{l}$ is some special channel ϵ that does not belong to any synchronisation set $\mathtt{cs}$.