Linearization of CIF through SOS

Linearization of CIF Through SOS

D.E. Nadales Agut M.A. Reniers
Systems Engineering
Department of Mechanical Engineering
Eindhoven University of Technology (TU/e)
{d.e.nadales.agut, m.a.reniers}@tue.nl

Linearization is the procedure of rewriting a process term into a linear form, which consist only of basic operators of the process language. This procedure is interesting both from a theoretical and a practical point of view. In particular, a linearization algorithm is needed for the Compositional Interchange Format (CIF), an automaton based modeling language.

The problem of devising efficient linearization algorithms is not trivial, and has been already addressed in literature. However, the linearization algorithms obtained are the result of an inventive process, and the proof of correctness comes as an afterthought. Furthermore, the semantic specification of the language does not play an important role on the design of the algorithm.

In this work we present a method for obtaining an efficient linearization algorithm, through a step-wise refinement of the SOS rules of CIF. As a result, we show how the semantic specification of the language can guide the implementation of such a procedure, yielding a simple proof of correctness.

1 Introduction

Linearization is the procedure of rewriting a process term into a linear form, which consist only of *basic operators* of a process language [10, 4, 15]. Linearization is also referred to as *elimination* in ACP style process algebras [1].

From a theoretical perspective, linearization of process terms is an interesting result. It allows to get a better understanding about the expressiveness of the language constructs, since it shows that all its terms are reducible to some normal form (which contains only a limited set of operators of the language). Also, linearization is useful in proving properties about closed terms, since the number of cases that needs to be dealt with in a proof by structural induction becomes smaller.

The Compositional Interchange Format (CIF) [2], is a language for modeling real-time, hybrid and embedded systems. CIF is developed to establish inter-operability of a wide range of tools by means of model transformations to and from the CIF. As such it plays a central role in the European projects Multiform [12], HYCON [9], C4C [5], and HYCON 2 [8]. CIF has a formal semantics [2], which is defined in terms of Structured Operational Semantics Rules (SOS) in the style of Plotkin [14].

Besides its theoretical importance, linearization of CIF models eliminates operators, such as urgency, that cannot be handled in other languages. Since CIF is meant to be used as an interchange format, the elimination of the operators broadens the set of models that can be translated to other languages. For the hierarchical extension of CIF [3], hCIF, linearization makes the elimination of hierarchy possible, and thus, all the tools available for CIF become available for use with hCIF models as well.

It is our goal to build a linearization algorithm for CIF, which results in an efficient representation of the original model, and such that all the operators of the language, such as parallel composition or synchronization are eliminated. The problem of efficient linearization has been already studied in literature [15, 4, 10] for process-algebraic languages for describing and analyzing discrete-event systems and hybrid systems. However, in the previous cases, the linearization algorithm is the result of an inventive process, and the proof of correctness comes as an afterthought. The semantic specification of the language does not play an important role on the design of the algorithm.

Previously, we studied the problem of implementing a simulator from the SOS specification of CIF [13]. The semantics of CIF is defined in terms of SOS rules, which induce a hybrid transition system, where each state contains a CIF term followed by a valuation (assignment of values to variables). This kind of semantics, even though useful for specification purposes, was not suitable for the implementation of a simulator (interpreter) for the language. This problem was solved by giving a set of SOS rules, called *symbolic rules*, which induced transition systems that do not contain the valuation part. It was also noted that the symbolic transition system induced by these rules is finite, and it resembles a (CIF) automaton. Thus, the symbolic SOS rules for CIF offer a straightforward algorithm for linearizing CIF models. However, the resulting automaton has a size that may be exponential in the size of the input model.

In this work we study the possibility of reusing the existing results on efficient linearization algorithms for obtaining a linear form of CIF from SOS rules. The idea is to give a more concrete version of the symbolic SOS rules of CIF (which is in turn a concrete version of the SOS rules with data), such that the transition system they induce can be translated to an automaton whose size does not grow exponentially as the result of interleaving actions (for synchronizing action the growth is still exponential, but in practice this is not a serious limitation since synchronization takes place only among a limited number of components).

As a result, we show a linearization procedure, which is obtained from the SOS specification of the language. In this way, the design of the algorithm requires less invention steps, reducing the opportunities to introduce mistakes, and at the same time it yields a simple proof of correctness.

2 Setting the Scene

For the discussion presented here, we consider a simplified version of CIF, which is untimed and contains only automata, a parallel composition operator, and a synchronizing action operator. This helps to keep the focus on the ideas, without distracting the reader with the complexity of CIF[1]. The techniques and results presented here can be easily extended to the setting of timed and hybrid systems, since we handle concepts such as invariants and time-can-progress conditions in a symbolic manner.

We begin by defining automata and the terms of our language. Throughout this work, notation $\mathscr{P}$ is used to refer to a set of predicates, $\mathscr{V}$ is a set of variables, $\mathscr{A}$ is a set of actions, τ is the silent action ($\tau \notin \mathscr{A}$), and $\mathscr{A}_\tau \triangleq \mathscr{A} \cup \{\tau\}$.

Definition 1 (Automaton) *An automaton is a tuple* $(V, \mathrm{init}, \mathrm{inv}, E, \mathrm{act}_S)$*, where* $V \subseteq \mathscr{L}$ *is a set of locations,* $\mathrm{init} \in V \to \mathscr{P}$ *is the initial predicate function,* $\mathrm{inv} \in V \to \mathscr{P}$ *is the invariant function,* $E \subseteq V \times \mathscr{A}_\tau \times \mathscr{P} \times V$ *is the set of edges, and* $\mathrm{act}_S \subseteq \mathscr{A}$ *is a set of synchronizing actions.*

Figure 1 presents a model of a railroad gate. It has two modes of operation (locations), closed and opened, denoted C and O respectively. Its initial predicate function associates the condition $wq = [\,]$ to location C (represented graphically with an incoming arrow without source location), and the predicate false to location O (represented by the absence of such an arrow). Here wq is the waiting queue that contains the id's of the trains waiting to pass through the gate, $[\,]$ is the empty list, and we denote lists by

[1]This language contains over 30 deduction rules

writing their elements between brackets, and separated by commas. Location C has $n = 0$ as invariant, where n is the numbers of trains crossing the gate, and location O has invariant $n \leq 1$. The automaton synchronizes with other components in actions *rq*, *go*, and *out*.

The automaton has four edges. Two edges $(C, rq, wq^+ = wq +\!\!+ [id^+], C)$, and $(O, rq, wq^+ = wq + \\ + [id^+], O)$, which are used to enqueue requests from the trains that want to pass the gate. Given two sequences *xs* and *ys*, $xs +\!\!+ ys$ denotes their concatenation. The predicate $wq^+ = wq +\!\!+ [id^+]$ expresses that the new value of the waiting queue after performing action *rq* will be the old waiting queue (*wq*) extended with the id of the train that request access (this id is contained in variable id^+). Graphically these edges are represented by two self loops in locations C and O, labeled $rq, wq^+ = wq +\!\!+ [id^+]$. The gate can make a transition from the closed state to the opened state, by issuing a *go* action, which sends the id at the front of the waiting queue using variable p.

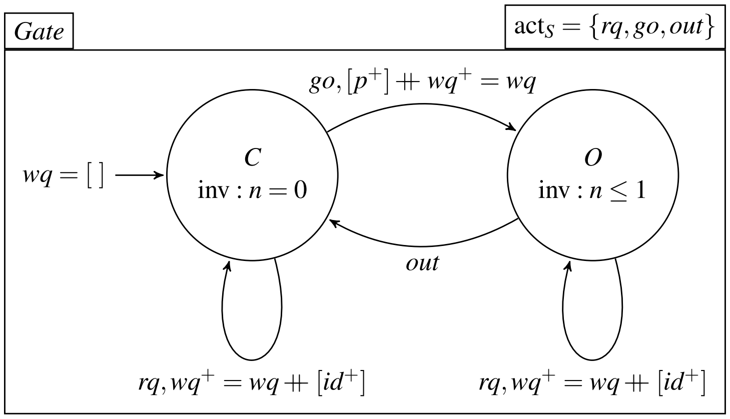

Gate
act$_S$ = {rq, go, out}
go, [p$^+$] ++ wq$^+$ = wq
wq = [ ]
C
inv : n = 0
O
inv : n ≤ 1
out
rq, wq$^+$ = wq ++ [id$^+$]
rq, wq$^+$ = wq ++ [id$^+$]

Figure 1: CIF model of a gate.

In Figure 2 we present the model of a train, which will be run in parallel with the gate model. It has a parameter i, which represents the train's id. It has four locations: far (F), near (N), stopped (S), and passing (P). Location F is the only initial location. When the train approaches the gate it issues a request to pass the gate by sending its id though variable *id*. Once in the near location, it can only go to the passing state if variable p is updated to its id (this update is carried out by the gate, as we have seen above). Otherwise it makes a transition to the stopped state. When the train enters the gate it increments variable n, and it decrements it upon departure.

These models can be composed in parallel using the parallel composition operator, denoted as $\|$. Actions in CIF are not synchronizing by default. Thus in the parallel composition

$$Train(0) \parallel Train(1)$$

the actions of the two trains will be interleaved.

We want to put the parallel composition of the two trains in parallel with the gate automaton, in such a way that the trains synchronize with the actions *rq*, *go*, and *out* of the gate. This can be achieved using the *synchronizing action* operator, denoted as γ_A. Informally, composition $\gamma_A(p)$ behaves as composition p, except that all the actions of the set A are made synchronizing in p. Below we explain this. Using these operators, we can express train gate model in CIF as follows:

$$\gamma_{\{rq,go,out\}}(Train(0) \parallel Train(1)) \parallel Gate \tag{1}$$

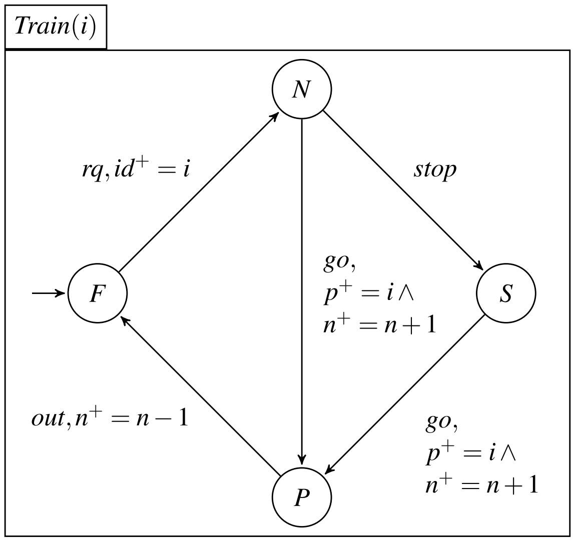


Figure 2: CIF model of a train.

As a consequence of the use of the synchronizing action operator in (1), action $i \in \{rq, go, out\}$ in *Train(j)*, $j \in \{0,1\}$, will synchronize with action i in the gate. Actions in the set $\{rq, go, out\}$ are interleaved in the parallel compositions of the trains (they *do not* synchronize) since the scope operator only make actions synchronizing in the outer scope. For more details see the rules of and their explanation Table 1.

Formally, the set of all CIF compositions is defined as follows:

Definition 2 (Compositions) *The set $\mathscr{C}$ of all compositions is defined through the following abstract grammar: $\mathscr{C} ::= \alpha \mid \mathscr{C} \parallel \mathscr{C} \mid \gamma_A(\mathscr{C})$, where α is an automaton and $A \subseteq \mathscr{A}$.*

In the next section we present the formal semantics of CIF compositions, both its explicit version and its symbolic counterpart.

2.1 Explicit and Symbolic Semantics of CIF

The semantics of CIF is defined in terms of hybrid transition systems [6]. In the context of the present work, we restrict our attention to ordinary transition systems (thus omitting time transitions), extended with *environment transitions* (see below).

The labeled transition systems we are considering have states of the form (p, σ). Here $p \in \mathscr{C}$, and $\sigma \in \Sigma$ is a valuation, where $\Sigma = \mathscr{V} \to \Lambda$, and Λ denotes a set of values. The valuation records the values of the model variables at a certain moment. There are two types of transitions in these labeled transition systems. *Action transitions*, of the form

$$(p, \sigma) \xrightarrow{a,b} (p', \sigma')$$

model the execution of an action a by composition p in an initial valuation σ, which changes composition p into p' and results in a new valuation σ'. Label b is a boolean that indicates whether action a is synchronizing. *Environment transitions*, of the form

$$(p, \sigma) \overset{A}{\dashrightarrow} (p', \sigma')$$

model the fact that the initial conditions and invariants of p (p' respectively) are satisfied in σ (σ'), and A is the set of synchronizing actions of p and p'. Environment transitions are used to obtain the state changes allowed by a model in a parallel composition context.

The transition system associated to a composition can be obtained by means of SOS rules. Below we present the explicit rules, where we have omitted the symmetric version of the parallel composition rule. Given a valuation σ, we define $\sigma'^{+} \triangleq \{(x^{+}, v) \mid (x, v) \in \sigma\}$. We use notation α to refer to the automaton $(V, \text{init}, \text{inv}, E, \text{act}_S)$, and $\alpha[x]$ to refer to $(V, \text{id}_x, \text{inv}, E, \text{act}_S)$, where $\text{id}_x(w) \triangleq w \equiv x$. Throughout this work, $\text{FV}(p)$ is the set of free variables of p.

$$\frac{\begin{array}{c}(v,a,r,v') \in E, \sigma \models \text{init}(v) \wedge \text{inv}(v),\\ \sigma' \models \text{inv}(v'), \sigma'^{+} \cup \sigma \models r,\\ \langle \forall x :: x^{+} \notin \text{FV}(r) \Rightarrow \sigma(x) = \sigma'(x) \rangle\end{array}}{(\alpha,\sigma) \xrightarrow{a, a \in \text{act}_S} (\alpha[v'],\sigma')}\;1
\qquad
\frac{\begin{array}{c}v \in V, \sigma \models \text{init}(v) \wedge \text{inv}(v),\\ \sigma' \models \text{inv}(v)\end{array}}{(\alpha,\sigma) \overset{\text{act}_S}{\dashrightarrow} (\alpha[v],\sigma')}\;2$$

$$\frac{(p,\sigma) \xrightarrow{a,\text{true}} (p',\sigma'), (q,\sigma) \xrightarrow{a,\text{true}} (q',\sigma')}{(p \parallel q,\sigma) \xrightarrow{a,\text{true}} (p' \parallel q',\sigma')}\;3
\qquad
\frac{(p,\sigma) \xrightarrow{a,b} (p',\sigma'), (q,\sigma) \overset{A}{\dashrightarrow} (q',\sigma'), a \notin A}{(p \parallel q,\sigma) \xrightarrow{a,b} (p' \parallel q',\sigma')}\;4$$

$$\frac{(p,\sigma) \overset{A_p}{\dashrightarrow} (p',\sigma'), (q,\sigma) \overset{A_q}{\dashrightarrow} (q',\sigma')}{(p \parallel q,\sigma) \overset{A_p \cup A_q}{\dashrightarrow} (p' \parallel q',\sigma')}\;5
\qquad
\frac{(p,\sigma) \xrightarrow{a,b,X} (p',\sigma')}{(\gamma_A(p),\sigma) \xrightarrow{a, b \vee a \in A, X} (\gamma_A(p'),\sigma')}\;6$$

$$\frac{(p,\sigma) \overset{A'}{\dashrightarrow} (p',\sigma')}{(\gamma_A(p),\sigma) \overset{A \cup A'}{\dashrightarrow} (\gamma_A(p'),\sigma')}\;7$$

Table 1: Explicit rules for CIF

Rule 1 states that an action can be triggered by an automaton, if there is an edge (v,a,r,v') such that the initial predicate and the invariant are satisfied in the initial valuation σ, and it is possible to find a new valuation σ' in which the invariant and the reset predicate are satisfied. The only variables that change in σ' w.r.t. σ are those free variables of r that are of the form x^{+}. Rule 2 states that an automaton is consistent in initial valuation σ if the initial predicate and invariant are satisfied in σ, and the valuation can be changed to σ' only if the invariant is preserved. Rule 3 expresses that an action a can be executed synchronously if it is *marked as synchronizing* in both components. The interleaving behavior is modeled in Rule 4, where an action a can be executed in p if *it is not synchronizing* in q. In Rule 6 an action a is marked as synchronizing if $a \in A$, or a is synchronizing in p. The environment rule for the synchronizing action operator (Rule 7) adds A to the set of synchronizing actions of p.

As noted in [13], the explicit rules are not suitable for implementation purposes. These rules often induce infinitely branching transition systems, and as a consequence it is not possible to obtain the set of possible successor states. In particular, the labels of the hybrid transition systems contain *trajectories*, of an dense domain, which are defined in the rules through computations over these dense sets. Another problem is that the valuations specify implicit constraints, such as "variables owned by a certain automaton cannot be changed in a parallel composition", which require to compute operations on infinite sets of valuations to get the set of possible successor states.

The solution to the problem explained above was to obtain a set of *symbolic rules* [7] from the explicit SOS specification. These symbolic rules represent the possible state changes by means of predicates, and thus, the state change caused by an action is visible on the arrows of the transitions. The symbolic rules for the language considered in this paper are shown in Table 2.

$$\frac{(v,a,r,v')\in E}{\langle\alpha\rangle \xrightarrow{a,a\in\mathrm{act}_S,\mathrm{init}(v),\mathrm{inv}(v),\mathrm{inv}(v'),r} \langle\alpha[v']\rangle}\ 8 \qquad \frac{v\in V}{\langle\alpha\rangle \overset{\mathrm{init}(v),\mathrm{inv}(v),\mathrm{act}_S}{\dashrightarrow} \langle\alpha[v]\rangle}\ 9$$

$$\frac{\langle p\rangle \xrightarrow{a,\mathrm{true},u_p,n_p,n'_p,r_p} \langle p'\rangle, \langle q\rangle \xrightarrow{a,\mathrm{true},u_q,n_q,n'_q,r_q} \langle q'\rangle}{\langle p \parallel q\rangle \xrightarrow{a,\mathrm{true},u_p\wedge u_q,n_p\wedge n_q,n'_p\wedge n'_q,r_p\wedge r'_p} \langle p' \parallel q'\rangle}\ 10 \qquad \frac{\langle p\rangle \xrightarrow{a,b,u_p,n_p,n'_p,r} \langle p'\rangle, \langle q\rangle \overset{u_q,n_q,A}{\dashrightarrow} \langle q'\rangle, a\notin A}{\langle p \parallel q\rangle \xrightarrow{a,b,u_p\wedge u_q,n_p\wedge n_q,n'_p\wedge n_q,r} \langle p' \parallel q'\rangle}\ 11$$

$$\frac{\langle p\rangle \overset{u_p,n_p,A_p}{\dashrightarrow} \langle p'\rangle, \langle q\rangle \overset{u_q,n_q,A_q}{\dashrightarrow} \langle q'\rangle}{\langle p \parallel q\rangle \overset{u_p\wedge u_q,n_p\wedge n_q,A_p\cup A_q}{\dashrightarrow} \langle p' \parallel q'\rangle}\ 12 \qquad \frac{\langle p\rangle \xrightarrow{a,b,u,n,n',r} \langle p'\rangle}{\langle\gamma_A(p)\rangle \xrightarrow{a,b\vee a\in A,u,n,n',r} \langle\gamma_A(p')\rangle}\ 13$$

$$\frac{\langle p\rangle \overset{u,n,A'}{\dashrightarrow} \langle p'\rangle}{\langle\gamma_A(p)\rangle \overset{u,n,A\cup A'}{\dashrightarrow} \langle\gamma_A(p')\rangle}\ 14$$

Table 2: Symbolic rules for CIF

The explicit and symbolic rules are related by the following soundness and completeness theorems. These theorems state how an explicit transition system can be reconstructed from its symbolic version, and vice-versa.

Theorem 1 (Soundness of action transitions) *For all p, p', a, b, u, n, n', r, σ, and σ' we have that if the following conditions hold:*

1. $\langle p\rangle \xrightarrow{a,b,u,n,n',r} \langle p'\rangle$
2. $\sigma \models u$, $\sigma \models n$, $\sigma' \models n'$, *and* $\sigma'^{+} \cup \sigma \models r$
3. $\langle \forall x :: x^{+} \notin FV(r) \Rightarrow \sigma(x) = \sigma'(x)\rangle$

then, there is a explicit action transition $(p,\sigma) \xrightarrow{a,b} (p',\sigma')$.

Theorem 2 (Completeness of action transitions) *For all p, p', a, b, σ, and σ' we have that if there is a explicit transition $(p,\sigma) \xrightarrow{a,b} (p',\sigma')$ then there exists u, n, n', and r such that the following conditions hold:*

1. $\langle p\rangle \xrightarrow{a,b,u,n,n',r} \langle p'\rangle$
2. $\sigma \models u$, $\sigma \models n$, $\sigma' \models n'$, *and* $\sigma'^{+} \cup \sigma \models r$
3. $\langle \forall x :: x^{+} \notin FV(r) \Rightarrow \sigma(x) = \sigma'(x)\rangle$

Theorem 3 (Soundness of environment transitions) *For all p, p', u, A, σ, and σ' we have that if the following conditions hold:*

1. $\langle p\rangle \overset{u,n,A}{\dashrightarrow} \langle p'\rangle$

2. $\sigma \models u$, $\sigma \models n$, $\sigma' \models n$

then, there is a explicit environment transition $(p,\sigma) \overset{A}{\dashrightarrow} (p',\sigma')$.

Theorem 4 (Completeness of environment transitions) *For all* p, p', A, σ, *and* σ' *we have that if there is an explicit transition* $(p,\sigma) \overset{A}{\dashrightarrow} (p',\sigma')$ *then there exists* u, *and* n *such that the following conditions hold:*

1. $\langle p\rangle \overset{u,n,A}{\dashrightarrow} \langle p'\rangle$

2. $\sigma \models u$, $\sigma \models n$, $\sigma' \models n$

It is not hard to see that given a CIF composition, the symbolic rules induce a *finite transition system*. For the model of the train gate presented in Section 2, a part of its associated symbolic transition system is shown in Figure 3 (the whole transition system contains 16 states), where we use the convention that for all x, y, z:

$$\langle x,y,z\rangle \equiv \gamma_{\{rq,go,out\}}(Train(0)[x] \parallel Train(1)[y]) \parallel Gate[z]$$

In this transition system, two problems can be noted. The size of the symbolic transition system grows exponentially as more trains are added. This is the result of the interleaving actions that are executed between these models. Secondly, there is a great deal of *redundant information*. The invariants of the source and the target states are present not only in the labels of action transitions, but also in the environment transition of these states. Similarly, the initialization conditions are meaningful only for the initial environment transition. For the remaining environment transitions in the systems, the initialization predicate is always true. In the next section we show how to overcome these problems using a new kind of symbolic rules.

3 Linear Transition Systems

In this section we define a structure called *linear transition system* (LiTS), which contains all the information necessary to represent any arbitrary CIF composition, and that can be translated to an equivalent automaton.

Consider the symbolic transition system of the train gate model. In Figure 3, we show a transition of the form:

$$\langle N,F,C\rangle \xrightarrow{go,p^+=0\wedge n^+=n+1\wedge wq^+ +\!\!\!+\ [p^+]=wq,n=0,n\leq 1} \langle P,F,O\rangle$$

The complete symbolic transition system also contains these transitions:

$$\langle N,N,C\rangle \xrightarrow{go,p^+=0\wedge n^+=n+1\wedge wq^+ +\!\!\!+\ [p^+]=wq,n=0,n\leq 1} \langle P,N,O\rangle$$
$$\langle N,S,C\rangle \xrightarrow{go,p^+=0\wedge n^+=n+1\wedge wq^+ +\!\!\!+\ [p^+]=wq,n=0,n\leq 1} \langle P,S,O\rangle$$

These three transitions only differ in the second component of the symbolic state, that is, the location in which the second train is. However, this information is not relevant for computing the state change. If we replace the above transitions by a unique transition of the form:

$$\langle N,\_,C\rangle \xrightarrow{go,p^+=0\wedge n^+=n+1\wedge wq^+ +\!\!\!+\ [p^+]=wq,n=0,n\leq 1} \langle P,\_,O\rangle$$

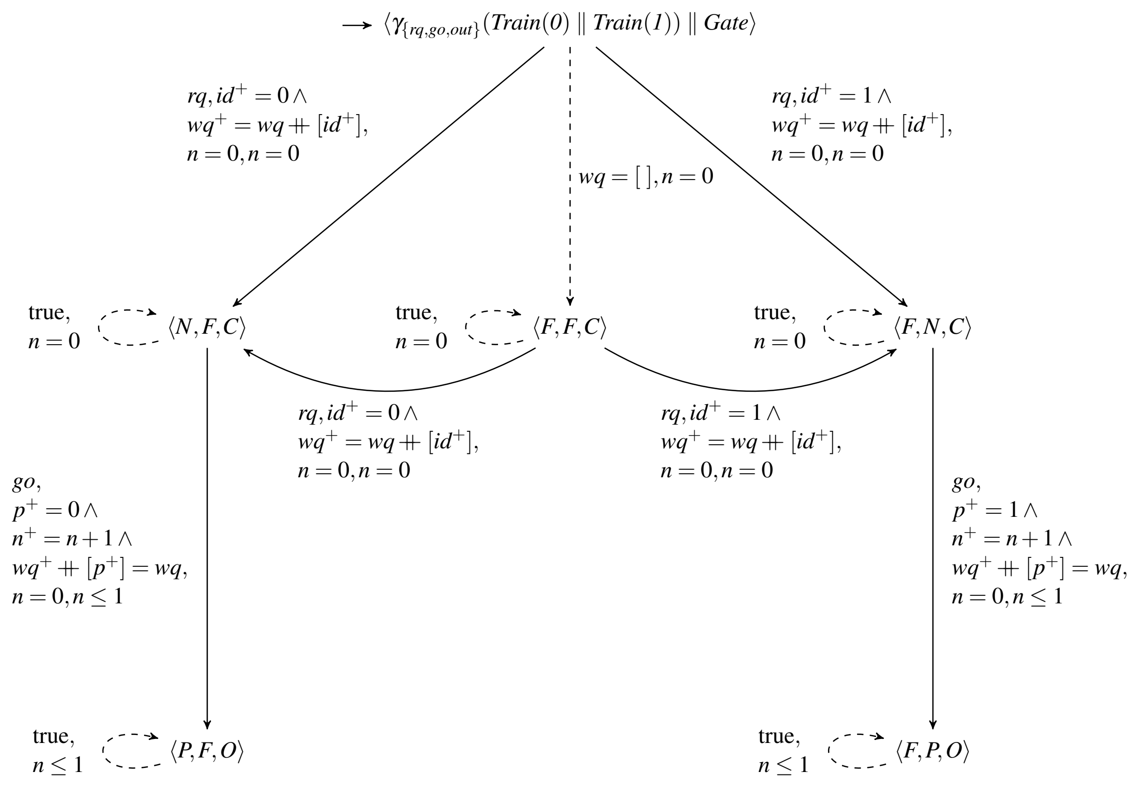

$\langle\gamma_{\{rq,go,out\}}(Train(0) \parallel Train(1)) \parallel Gate\rangle$
$rq, id^+ = 0 \wedge$ $wq^+ = wq ++ [id^+]$, $n = 0, n = 0$
$wq = [\,], n = 0$
$rq, id^+ = 1 \wedge$ $wq^+ = wq ++ [id^+]$, $n = 0, n = 0$
true, $n = 0$
$\langle N, F, C\rangle$
true, $n = 0$
$\langle F, F, C\rangle$
true, $n = 0$
$\langle F, N, C\rangle$
$rq, id^+ = 0 \wedge$ $wq^+ = wq ++ [id^+]$, $n = 0, n = 0$
$rq, id^+ = 1 \wedge$ $wq^+ = wq ++ [id^+]$, $n = 0, n = 0$
go, $p^+ = 0 \wedge$ $n^+ = n + 1 \wedge$ $wq^+ ++ [p^+] = wq$, $n = 0, n \leq 1$
go, $p^+ = 1 \wedge$ $n^+ = n + 1 \wedge$ $wq^+ ++ [p^+] = wq$, $n = 0, n \leq 1$
true, $n \leq 1$
$\langle P, F, O\rangle$
true, $n \leq 1$
$\langle F, P, O\rangle$

Figure 3: A part of the symbolic transition system for the train gate controller

then we can avoid the state explosion caused by the interleaving actions. Here the *wild-card* symbol $\_$ can be read as "for any location".

Furthermore, in Figure 3 we see that there is no need to replicate the entire structure in a given transition, since it suffices to keep track of the locations that change.

From the observation above, we want a linear transition system where the states are *sequences of locations*, containing also *wild-cards*. These wild-cards are used to denote the fact that the location of a certain automaton does not change in the transition. Formally the states of the LiTS belong to the set

$$(\mathscr{L} \times \{\_\})^* \tag{2}$$

where $\_$ is the wild-card symbol, and A^* is the set of all sequences whose elements are taken from the set A. An example of such state is the list $[F, \_, C]$.

The next thing to define is the transitions of the LiTS's, in such a way that the redundancy introduced by the STS's is eliminated. To accomplish this, we split action and environment transitions into several transitions, which are described next.

Action Transitions They are of the form $p \models \langle vs\rangle \xrightarrow{a,r} \langle vs'\rangle$, where $p \in \mathscr{C}$ is a composition, $a \in \mathscr{A}_\tau$ is an action label, and $r \in \mathscr{P}$ is the update predicate associated to the action.

Synchronizing Actions They are of the form $p \overset{\text{sync}}{\rightsquigarrow} A$, where $p \in \mathscr{C}$ is a composition, and $A \subseteq \mathscr{A}$ is the set of synchronizing actions of p.

Initialization Transitions They are of the form $p \overset{\text{ipred}}{\rightsquigarrow} fs$, where $p \in \mathscr{C}$ and $fs \in (\mathscr{L} \rightharpoonup \mathscr{P})^*$ is a list containing the initialization predicate function of each automaton in p.

Invariant Transitions They are of the form $p \overset{\text{inv}}{\rightsquigarrow} fs$, where $p \in \mathscr{C}$ is a composition, and $fs \in (\mathscr{L} \to \mathscr{P})^*$ is a list containing the invariant function associated to each automaton in p.

The reader may have expected initialization or invariant transitions of the form:

$$vs \overset{\text{inv}}{\rightsquigarrow} p$$

where vs is a list of locations, and p is a predicate. However this approach requires enumerating the state space explicitly to construct the $\overset{\text{inv}}{\rightsquigarrow}$ relation. By using lists of functions we avoid this explicit construction.

Wild-card Transitions They are of the form $p \overset{\#}{\rightsquigarrow} xs$, where $p \in \mathscr{C}$ is a composition, and $xs \in (\_)^*$ is a sequence of wild-cards whose size coincides with the number of automata that are composed in parallel in p. These transition are not needed for reconstructing the environment transitions, they are used in the linear SOS rules to model the fact that nothing changes in a component of a parallel composition, when the other component performs an action.

In Table 3 we show some of the linear SOS rules for CIF compositions. We have omitted the rules for synchronizing actions, initialization, and wild-card transitions since they are similar to the invariant transitions.

The linear rules can be easily to obtained from the symbolic ones. For action rules, invariants and initialization predicates, and the synchronizing action label are simply omitted (since they can be obtained from other transitions). The linear rule for interleaving parallel composition is almost identical to the symbolic rule. The only differences are that the set A is obtained from a $\overset{\text{sync}}{\rightsquigarrow}$ transition, and we use the wild-card transition to represent the fact that the locations of the other automaton are not relevant (at the symbolic level at least). A similar observation can be made for the rule for parallel composition. In this case since we do not have the synchronizing label, we reconstruct it from the $\overset{\text{sync}}{\rightsquigarrow}$ transition. This label is equivalent to $a \in A$, thus a label true in both components is equivalent to $a \in A_p \wedge a \in A_q$, which is in turn equivalent to $a \in A_p \cap A_q$.

$$\frac{}{(V, \text{init}, \text{inv}, \text{tcp}, E, \text{act}_S) \overset{\text{inv}}{\rightsquigarrow} [\text{inv}]}\ 15 \qquad \frac{p \overset{\text{inv}}{\rightsquigarrow} fs_p, q \overset{\text{inv}}{\rightsquigarrow} fs_q}{p \parallel q \overset{\text{inv}}{\rightsquigarrow} fs_p +\!\!+ fs_q}\ 16$$

$$\frac{(v, a, r, v') \in E}{(V, \text{init}, \text{inv}, \text{tcp}, E, \text{act}_S) \models \langle [v] \rangle \xrightarrow{a,r} \langle [v'] \rangle}\ 17 \qquad \frac{p \models \langle vs \rangle \xrightarrow{a,r} \langle vs' \rangle, q \overset{\text{sync}}{\rightsquigarrow} A, q \overset{\#}{\rightsquigarrow} \_, a \notin A,}{p \parallel q \models \langle vs +\!\!+ \_ \rangle \xrightarrow{a,r} \langle vs' +\!\!+ \_ \rangle}\ 18$$

$$\frac{p \models \langle vs_p \rangle \xrightarrow{a,r_p} \langle vs'_p \rangle, q \models \langle vs_q \rangle \xrightarrow{a,r_q} \langle vs'_q \rangle, p \overset{\text{sync}}{\rightsquigarrow} A_p, q \overset{\text{sync}}{\rightsquigarrow} A_q, a \in A_p \cap A_q}{p \parallel q \models \langle vs_p +\!\!+ vs_q \rangle \xrightarrow{a,r_p \wedge r_q} \langle vs'_p +\!\!+ vs'_q \rangle}\ 19$$

$$\frac{p \overset{\text{inv}}{\rightsquigarrow} fs}{\gamma_A(p) \overset{\text{inv}}{\rightsquigarrow} fs}\ 20 \qquad \frac{p \models \langle vs \rangle \xrightarrow{a,r} \langle vs' \rangle}{\gamma_A(p) \models \langle vs \rangle \xrightarrow{a,r} \langle vs' \rangle}\ 21$$

Table 3: Linear SOS rules for CIF compositions

If a composition p contains no synchronizing actions, then the size of its induced transition system is linear w.r.t. the size of p. However, the size of the LiTS also depends on the number of synchronizing actions. The following property gives the formal details.

Property 1 (Size of the linear transition system) *Let p be a CIF composition, such that it contains n automata $\alpha_i \equiv (V_i, \mathrm{init}_i, \mathrm{inv}_i, E, \mathrm{act}_{S_i})$, $0 \leq i < n$. Let a be the only synchronizing action in these automata. Then the number of transitions in the LiTS associated to p is given by:*

$$\sum_{0\leq i<n} \#\{x \mid (v,g,x,u,v') \in E_i \wedge x \neq a\} + \prod_{0\leq i<n} \#\{x \mid (v,g,x,u,v') \in E_i \wedge x = a\} \tag{3}$$

where $\#A$ is the number of elements in set A.

In spite of the fact that the number in (3) can be significantly large, in practice, communication among components is usually restricted to a few automata, and the number of edges of an automaton that contain a given synchronizing action a is small.

3.1 Relating LiTS and STS

In the same way symbolic transitions are related to explicit ones via soundness and completeness results, linear transitions have the same property w.r.t. symbolic transitions.

The first two results state that a LiTS contains all the necessary information to reconstruct the environment transitions in the symbolic transition system and vice-versa. Here "leads to transitions" refers to the initialization, invariant, and synchronizing actions transitions in the LiTS. Given a composition p, which contains n atomic automata, and a sequence ls of n locations, $p[ls]$ is the composition obtained by replacing the initial predicate function of the i^{th} automaton by $\mathrm{id}_{ls.i}$, for $0 \leq i < n$, where $ls.i$ is the element of sequence ls at position i (sequences are numbered starting from 0). $\mathrm{locsof}(p)$ refers to the set of sequences ls, where $\#ls = n$ and $ls.i$ is a location of the i^{th} automaton of composition p ($0 \leq i < n$).

Theorem 5 (Soundness of leads to transitions) *For all p, is, fs, gs, A, u, and n we have that if the following conditions hold:*

1. $is \in \mathrm{locsof}(p)$
2. $p \overset{\mathrm{ipred}}{\rightsquigarrow} fs$, $p \overset{\mathrm{inv}}{\rightsquigarrow} gs$, $p \overset{\mathrm{sync}}{\rightsquigarrow} A$
3. $u = \bigwedge_{0\leq i<\#fs} fs.i(is.i)$, *and* $n = \bigwedge_{0\leq i<\#gs} gs.i(is.i)$

then there is a symbolic transition $\langle p\rangle \overset{u,n,A}{\dashrightarrow} \langle p[is]\rangle$.

Theorem 6 (Completeness of leads to Transitions) *For all p, u, n, A, and p' we have that if there is an environment transition $\langle p\rangle \overset{u,n,A}{\dashrightarrow} \langle p'\rangle$ then there are is, fs, gs, u, and n such that the following conditions hold:*

1. $is \in \mathrm{locsof}(p)$
2. $p \overset{\mathrm{ipred}}{\rightsquigarrow} fs$, $p \overset{\mathrm{inv}}{\rightsquigarrow} gs$, $p \overset{\mathrm{sync}}{\rightsquigarrow} A$
3. $u = \bigwedge_{0\leq i<\#fs} fs.i(is.i)$, *and* $n = \bigwedge_{0\leq i<\#gs} gs.i(is.i)$, $p' \equiv p[is]$

The soundness theorem for linear action transitions shows how a symbolic action transition can be obtained, using the leads to transitions as well. Functions $\sqsubseteq$ and $\succ$ are defined below, where $x : xs$ is the list that results after appending the element x to the front of xs.

Definition 3 (Sub-sequence and sequence overwriting) *Function $\sqsubseteq \in A^* \rightharpoonup A^* \rightharpoonup \mathbb{B}$ is defined as follows:*

$$[\,] \sqsubseteq xs \triangleq \text{true}$$
$$(x : xs) \sqsubseteq (y : ys) \triangleq ((x \equiv \_) \vee (x \equiv y)) \wedge xs \sqsubseteq ys$$

Function $\succ \in A^* \rightharpoonup A^* \rightharpoonup A^*$ is defined as follows:

$$[\,] \succ xs \triangleq xs$$
$$(x : xs) \succ (y : ys) \triangleq \begin{cases} x : (xs \succ ys) & \text{if } x \neq \_ \\ y : (xs \succ ys) & \text{if } x = \_ \end{cases}$$

Theorem 7 (Soundness of Linear Action Transitions) *For all p, vs, a, r, vs', is, fs, gs, u, n, n, and A we have that if the following conditions hold:*

1. $is \in \text{locsof}(p)$
2. $p \models \langle vs \rangle \xrightarrow{a,r} \langle vs' \rangle$, $p \overset{\text{ipred}}{\rightsquigarrow} fs$, $p \overset{\text{inv}}{\rightsquigarrow} gs$, $p \overset{\text{sync}}{\rightsquigarrow} A$
3. $u = \bigwedge_{0 \leq i < \#fs} fs.i(is.i)$, *and* $n = \bigwedge_{0 \leq i < \#gs} gs.i(is.i)$, $n' = \bigwedge_{0 \leq i < \#gs} gs.i((vs' \succ is).i)$, $vs \sqsubseteq is$, $b \equiv a \in A$

then there is a symbolic transition: $\langle p \rangle \xrightarrow{a,b,u,n,n',r} \langle p[vs' \succ is] \rangle$.

Theorem 8 (Completeness of Linear Action Transitions) *For all p, p', a, b, u, n, n', and r we have that if there is a symbolic transition:*

$$\langle p \rangle \xrightarrow{a,b,u,n,n',r} \langle p' \rangle$$

then there are vs, vs', is, fs, gs, and A such that the following conditions hold:

1. $is \in \text{locsof}(p)$
2. $p \models \langle vs \rangle \xrightarrow{a,r} \langle vs' \rangle$, $p \overset{\text{ipred}}{\rightsquigarrow} fs$, $p \overset{\text{inv}}{\rightsquigarrow} gs$, $p \overset{\text{sync}}{\rightsquigarrow} A$
3. $u = \bigwedge_{0 \leq i < \#fs} fs.i(is.i)$, *and* $n = \bigwedge_{0 \leq i < \#gs} gs.i(is.i)$, $n' = \bigwedge_{0 \leq i < \#gs} gs.i((vs' \succ is).i)$, $vs \sqsubseteq is$, $b \equiv a \in A$, $p' \equiv p[vs' \succ is]$

These theorems can be proved using structural induction. The proofs are relatively simple, and are omitted due to space constraints.

4 Obtaining a Linear Automaton from a LiTS

Once a linear transition system is induced by the SOS rules, we need a way to obtain a linear automaton from it. In this section we describe the procedure, and we show that the generated automaton is stateless bisimilar [11] to the composition that induced the transition system. Both from a theoretical and a practical point of view this is an interesting result, which tells us that every composition can be reduced to an automaton (this is intuitively obvious for the language we present here, but it is not for CIF and its hierarchical extension).

Formally, given an composition p and its associated LiTS M, we want to build an automaton α_p such that p has the same behavior as α_p. The idea is to simulate the execution of M, using α_p. To this end, we need to introduce a sequence of variables ls, which are used to represent the active state in M in a given execution. We call these variables *location pointers* [10]. Below, we give the definition[2] of the linearization function, which returns the automaton associated to a given composition and the location pointers used in it. The second component returned by the function is used later to formulate the correctness result.

Definition 4 (Linearization Function) *Let p be a CIF composition. Function $\mathrm{L} \in \mathscr{C} \to (\mathscr{C} \times \mathscr{V}^*)$ is defined as the least function that satisfies:*

$$\mathrm{L}(p) = ((\{x\}, \mathrm{init}, \mathrm{inv}, E, \mathrm{act}_S), ls)$$

where

- $p \overset{\#}{\rightsquigarrow} xs$, $\#xs = n$, $p \overset{\mathrm{ipred}}{\rightsquigarrow} fs$, $p \overset{\mathrm{inv}}{\rightsquigarrow} gs$, $p \overset{\mathrm{sync}}{\rightsquigarrow} A$
- $\langle \forall i :: 0 \leq i < n \Rightarrow ls.i \notin \mathrm{FV}(p) \rangle$, $x \in \mathscr{L}$
- $\mathrm{init}(x) = (\bigwedge_{0 \leq i < n} \bigwedge_{v \in \mathrm{dom}(fs.i)} (ls.i = v \Rightarrow fs.i(v))) \wedge (\bigwedge_{0 \leq i < n} ls.i \in \mathrm{dom}(fs.i))$
- $\mathrm{inv}(x) = \bigwedge_{0 \leq i < n} \bigwedge_{v \in \mathrm{dom}(gs.i)} (ls.i = v \Rightarrow gs.i(v))$
- $E = \{(x, a, r \wedge \bigwedge_{\substack{0 \leq i < n \\ vs.i \neq \_}} ls.i = vs.i \wedge ls.i^+ = vs'.i, x) \mid p \models \langle vs \rangle \xrightarrow{a,r} \langle vs' \rangle\}$

In the above definition we introduce n free variables, which are used as location pointers, and we use a location x (which can be defined as the least location in $\mathscr{L}$) as the unique location of the automaton. The initial predicate and invariant functions are conditional expressions, which ensure that the right predicate is chosen according to the values of the location pointers. In the definition of the init function, the second part of the conjunction forces the choice of an initial location (otherwise this predicate can be trivially satisfied). The set of edges is constructed from the action transition of the linear transition system. The reset mapping in the action transitions is extended with updates to the location pointers to keep track of the state in the linear transition system.

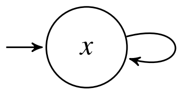
The well-definedness of function L is a consequence of the finiteness of LiTSs. Given a composition p, such that $\mathrm{L}(p) = (\alpha_p, ls)$, we say that α_p is the linear automaton associated to it.

For the train gate model, the linear automaton associated to it is shown in Figure 4, where the initial predicate and invariant functions are (once they are *simplified*):

$$\mathrm{init}(x) = (l_0 = F \wedge l_1 = F \wedge l_2 = C \wedge wq = [\,])$$
$$\mathrm{inv}(x) = (l_2 = C \Rightarrow n = 0) \wedge (l_2 = O \Rightarrow n \leq 1)$$

[2]Strictly speaking, function L is not uniquely determined, since it is possible to pick different location pointers. This can be avoided by defining a function that returns the least n fresh variables in a given composition (assuming variables are totally ordered). A similar observation can be done about location x.

Train Gate $\mathrm{act}_S = \{rq, go, out\}$



$rq, id^+ = 0 \wedge wq^+ = wq +\!\!\!+ [id^+] \wedge l_0 = F \wedge l_0^+ = N \wedge l_2 = C \wedge l_2^+ = C$

$stop, l_0 = N \wedge l_0^+ = S$

$go, p^+ = 0 \wedge n^+ = n + 1 \wedge wq^+ +\!\!\!+ [p^+] = wq \wedge l_0 = N \wedge l_0^+ = P \wedge l_2 = C \wedge l_2^+ = O$

$go, p^+ = 0 \wedge n^+ = n + 1 \wedge wq^+ +\!\!\!+ [p^+] = wq \wedge l_0 = S \wedge l_0^+ = P \wedge l_2 = C \wedge l_2^+ = O$

$out, n^+ = n + 1 \wedge l_0 = P \wedge l_0^+ = F \wedge l_2 = O \wedge l_2^+ = C$

$rq, id^+ = 1 \wedge wq^+ = wq +\!\!\!+ [id^+] \wedge l_1 = F \wedge l_1^+ = N \wedge l_2 = C \wedge l_2^+ = C$

$stop, l_1 = N \wedge l_1^+ = S$

$go, p^+ = 1 \wedge n^+ = n + 1 \wedge wq^+ +\!\!\!+ [p^+] = wq \wedge l_1 = N \wedge l_1^+ = P \wedge l_2 = C \wedge l_2^+ = O$

$go, p^+ = 1 \wedge n^+ = n + 1 \wedge wq^+ +\!\!\!+ [p^+] = wq \wedge l_1 = S \wedge l_1^+ = P \wedge l_2 = C \wedge l_2^+ = O$

$out, n^+ = n + 1 \wedge l_1 = P \wedge l_1^+ = F \wedge l_2 = O \wedge l_2^+ = C$

Figure 4: Linear version of the gateway model

The next step is to prove that the linear version of a composition is indeed equivalent to it. If we consider the transition systems they induce, we find that these differ significantly among each other: they have different labels, invariants, etc. Thus if we want to prove equivalence at the LiTS level, we need a non-trivial definition of equivalence.

A better strategy is to prove equivalence at the labeled transition level (using the explicit semantics). Therefore, we prove that p and its associated linear automaton are stateless bisimilar, after abstracting away the values of the program counters (or location pointers). The standard notion of strong bisimilarity [11] is defined below.

Definition 5 (Strong Bisimilarity for SOS) *A symmetric relation R is a strong bisimulation relation if for all $(p,q) \in R$, and for all σ, ℓ, p', σ' the following transfer conditions hold:*

1. $(p,\sigma) \xrightarrow{\ell} (p',\sigma') \Rightarrow \langle \exists q' :: (q,\sigma) \xrightarrow{\ell} (q',\sigma') \wedge (p',q') \in R \rangle$
2. $(p,\sigma) \overset{\ell}{\dashrightarrow} (p',\sigma') \Rightarrow \langle \exists q' :: (q,\sigma) \overset{\ell}{\dashrightarrow} (q',\sigma') \wedge (p',q') \in R \rangle$

Two closed terms p and q are strongly bisimilar, denoted $SOS \models p \leftrightarrow q$, if $(p,q) \in R$ for some strong bisimulation relation R.

Next, we present the SOS rules for the variable scope operator in Table 4 (the rules for environment transitions are similar and therefore omitted). In these rules we make use of the following notations:

- Given two sequences xs and ys, such that $\#xs = \#ys$, $\{xs \mapsto ys\} \in \mathrm{ran}(xs) \to \mathrm{ran}(ys)$ is a function defined as follows:
$$\{xs \mapsto ys\} = \{(xs.i, ys.i) \mid 0 \leq i < \#xs\}$$

- The notation above is overloaded to denote a similar function. We believe this keeps the notation concise and it does not bring confusion. Given a sequence xs and an element y, $\{xs \mapsto y\} \in \text{ran}(xs) \to \{y\}$ is a function defined as follows:

$$\{xs \mapsto y\} = \{(xs.i, y) \mid 0 \leq i < \#xs\}$$

- Symbol $\perp$ denotes the undefined value.

$$\frac{(p, \{xs \mapsto vs\} \succ \sigma) \xrightarrow{a,b} (p', \{xs \mapsto vs'\} \succ \sigma')}{(|[_{\text{V}}\, \{xs \mapsto vs\} :: p\ ]|, \sigma) \xrightarrow{a,b} (|[_{\text{V}}\, \{xs \mapsto vs'\} :: p'\ ]|, \sigma')}\ 22$$

$$\frac{(|[_{\text{V}}\, \{xs \mapsto vs\} :: p\ ]|, \sigma) \xrightarrow{a,b} (|[_{\text{V}}\, \{xs \mapsto vs'\} :: p'\ ]|, \sigma')}{(|[_{\text{V}}\, \{xs \mapsto \perp\} :: p\ ]|, \sigma) \xrightarrow{a,b} (|[_{\text{V}}\, \{xs \mapsto vs'\} :: p'\ ]|, \sigma')}\ 23$$

Table 4: SOS rules for the variable scope operator

Using the previously defined operator and the notion of stateless bisimilarity, we can enunciate the theorem which states that the linearization procedure is correct.

Theorem 9 (Correctness of the Linearization) *Let p be a composition, and $\text{L}(p) = (\alpha_p, ls)$. Then we have:*

$$SOS \models p \leftrightarrow |[_{\text{V}}\, \{ls \mapsto \perp\} :: \alpha_p\ ]|$$

Proof 1 *It is possible to prove that the following relation:*

$$\begin{aligned} R \triangleq &\{(p, |[_{\text{V}}\, \{ls \mapsto \perp\} :: \alpha_p\ ]|) \mid (\alpha_p, ls) = \text{L}(p)\} \cup \\ &\{(p[is], |[_{\text{V}}\, \{ls \mapsto is\} :: \alpha_p\ ]|) \mid (\alpha_p, ls) = \text{L}(p), is \in \text{locsof}(p)\} \end{aligned} \tag{4}$$

is a witness of the bisimulation. The proof uses the soundness and completeness results presented in Sections 2.1 and 3.1, and it does not require the use of structural induction.

5 Concluding Remarks

We have presented linearization algorithm for a subset of CIF, which shows that every CIF composition can be reduced to an automaton. The linearization procedure was obtained in a stepwise manner from the SOS specification of this language. In this way, SOS rules are used not only to specify the behavior of CIF, but also as a specification formalism for performing semantic preserving manipulations on the syntactic elements of the language.

The soundness and completeness results between the different transition systems give us a simple proof of correctness on the linearization procedure. The different levels in which a language is described (explicit, symbolic, and linear semantics) provide a convenient way to tackle specific problems. The explicit semantics is useful for achieving an abstract and succinct specification of the language. The symbolic semantics give us the means for specifying symbolic computations. Finally, the linear semantics yields an efficient representation of the state space associated to a given composition.

We conjecture the method presented here can be applied to any automaton based language. For process algebraic specification language it may not be suitable due to the presence of recursion.

As future work, we plan to extension the linearization algorithm to the full CIF, and therefore, to a hybrid setting. Time-can-progress predicates and dynamic types can be extracted in the same way invariants were extracted in this work, and therefore we expect no problems in this regard.

References

[1] J. C. M. Baeten, T. Basten & M. A. Reniers (2009): *Process Algebra: Equational Theories of Communicating Processes (Cambridge Tracts in Theoretical Computer Science)*, 1st edition. Cambridge University Press.

[2] J.C.M. Baeten, D.A. van Beek, D. Hendriks, A.T. Hofkamp, D.E. Nadales Agut, J.E. Rooda & R.R.H. Schiffelers (2010): *Definition of the Compositional Interchange Format*. Technical Report Deliverable D1.1.2, Multiform. Available at `www.multiform.bci.tu-dortmund.de/images/stories/multiform/deliverables/multiform_d112.pdf`.

[3] Harsh Beohar, Damian E. Nadales Agut, Dirk A. van Beek & Pieter J. L. Cuijpers (2010): *Hierarchical states in the Compositional Interchange Format*. Electronic Proceedings in Theoretical Computer Science 32, pp. 42–56, doi:10.4204/EPTCS.32.4.

[4] P. van de Brand, M. A. Reniers & P. J. L. Cuijpers (2006): *Linearization of Hybrid Processes*. Journal of Logic and Algebraic Programming 68(1–2), pp. 54–104, doi:10.1016/j.jlap.2005.10.003.

[5] C4C consortium (2008): *Control for Coordination of Distributed Systems*. `http://www.c4c-project.eu/`.

[6] P.J.L. Cuijpers & M.A. Reniers (2008): *Lost in Translation: Hybrid-Time Flows vs Real-Time Transitions*. In: HSCC 2008, Lecture Notes in Computer Science 4981, Springer, pp. 116–129, doi:10.1007/978-3-540-78929-1_9.

[7] M. Hennessy & H. Lin (1995): *Symbolic bisimulations*. In: Selected papers of the meeting on Mathematical foundations of programming semantics, Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands, pp. 353–389, doi:10.1016/0304-3975(94)00172-F. Available at `http://portal.acm.org/citation.cfm?id=202463.202370`.

[8] HYCON 2 consortium (2010): *Highly-complex and networked control systems*. `http://www.hycon2.eu/`.

[9] HYCON Network of Excellence (2005): *Hybrid Control: Taming Heterogeneity and Complexity of Networked Embedded Systems*. `http://www.ist-hycon.org/`.

[10] U. Khadim, D. A. van Beek & P. J. L. Cuijpers (2007): *Linearization of Hybrid Chi Using Program Counters*. Technical Report CS-Report 07-18, Eindhoven University of Technology, Department of Computer Science, The Netherlands.

[11] M. R. Mousavi, M. A. Reniers & J. F. Groote (2005): *Notions of bisimulation and congruence formats for SOS with data*. Information and Computation 200(1), pp. 107–147, doi:10.1016/j.ic.2005.03.002.

[12] MULTIFORM consortium (2008): *Integrated Multi-formalism Tool Support for the Design of networked Embedded Control Systems MULTIFORM*. `http://www.multiform.bci.tu-dortmund.de`.

[13] D. E. Nadales Agut & M. A. Reniers (2011): *Deriving a Simulator for a Hybrid Language Using SOS Rules*. `http://se.wtb.tue.nl/sewiki/cif/publications2`.

[14] Gordon D. Plotkin (2004): *A structural approach to operational semantics*. Journal of Logic and Algebraic Programming 60-61, pp. 17–139, doi:10.1016/j.jlap.2004.05.001.

[15] Yaroslav S. Usenko (2002): *Linearization in μCRL*. Ph.D. thesis, Eindhoven University of Technology.