Deriving a Simulator for a Hybrid Language Using SOS Rules [⋆]

D.E. Nadales Agut and M.A. Reniers

Systems Engineering
Department of Mechanical Engineering
Eindhoven University of Technology (TU/e)
d.e.nadales.agut@tue.nl m.a.reniers@tue.nl

Abstract. Notwithstanding the usefulness of Structured Operational Semantics (SOS) for theoretical aspects, its practical applications are limited in the context of hybrid languages. This kind of semantics specify important behavioral aspects in terms of computations over infinite objects but the implementation of tools requires a symbolic treatment of these computations. To derive a simulator from an SOS specification of a hybrid language, we make use of the symbolic SOS approach, obtaining a set of rules without data, which are proven to be a sound and complete representation of the original semantics, and at the same time they serve as the basis of the simulator. In this way, we show a practical application of SOS in the development of simulators for hybrid languages, which enables partial development of equational theories for the explicit version.

Keywords: semantics, hybrid-systems, formal methods

1 Introduction

In general, the behavior of hybrid languages (describing both discrete-event and continuous-time behavior) is specified through formal semantics [8,15,10]. This gives a precise meaning to specifications, provides a reference against which implementations can be judged, and makes the development of mathematical theories possible, enabling practical applications, such as model checking algorithms.

The Compositional Interchange Format (CIF)[3], is a language for modeling real-time, hybrid and embedded systems, whose purpose is to establish interoperability of a wide range of tools by means of model transformations to and from the CIF, playing a central role in the European projects Multiform [19], HYCON [12], and HYCON 2 [11].

CIF has a formal semantics based on the Structured Operational Semantics (SOS) framework [21], which is required for carrying out model preserving

[⋆] This work was partially supported by the European Community's Seventh Framework Programme (FP7/2007-2013) project MULTIFORM.

transformations from CIF to and from other formalisms (see [20] for an example). Using this kind of semantics, it is possible to make use of meta theory for establishing compositionality of the operators of the language [18], an essential property for an interchange format.

It is our interest to implement a simulator, based on the SOS rules of CIF. The ability of creating rapid prototypes from these specifications is crucial for making SOS useful because the impact of certain design decisions cannot be assessed until a running implementation is available. Additionally, it is through the use of the language in real world models that functional requirements, which cannot be mathematically captured, can be tested.

However, SOS rules with data are not suitable for a direct implementation of a simulator. These rules often induce infinitely branching transition systems, and as a consequence is not possible to obtain the set of possible successor states. In particular, the labels of the hybrid transition systems contain *trajectories*, of an infinite domain, which are defined in the rules through computations over these infinite objects. 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.

To obtain a set of SOS rules without data, called *symbolic SOS* [9] (SSOS), we represent the possible state changes by predicates. Then, we establish soundness and completeness results, which determine the relationship between the original SOS and its symbolic counterpart. In this way we show that the symbolic rules are a correct implementation of the original specification. In particular, the soundness results serve as the basis of the interpreter/simulator since they specify how to obtain transitions with data using the symbolic information.

The derivation of simulators from SOS semantics was considered already in [16]. In that work, the symbolic information needed for computing the next states could be obtained directly from the explicit SOS rules of the language. This is not the case for CIF, which presents concepts such as urgency and control variables whose symbolic treatment cannot be directly inferred from its semantics.

This work shows the application of formal methods to construct a simulator for a hybrid language from its SOS rules, in such a way that it is mathematically proved that it conforms to its original specification. To this end, following [14], we regard hybrid systems as computational executable artifacts: hybrid systems are a concurrent model of computation, and the "simulation" tools act as interpreters (or compilers) for these languages. Due to space constraints, in this paper we focus in a subset of CIF. Nevertheless, the results presented here are straightforward to extend to the complete language (since here we deal with the most complex part). The symbolic rules for the complete language and the proof of the theorems can be found in [23]. In this manner, we extend the work of Hennessy and Lin [9] to the setting of hybrid transition systems with data. Besides its use at the implementation level, we exhibit how symbolic SOS also enables the use of meta-theories for proving properties of the language, which are not available for SOS with data.

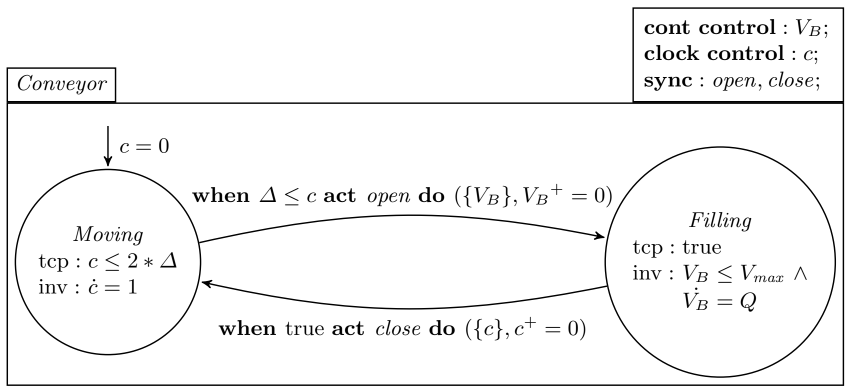
In Section 2 we present the syntax of CIF and the specification of its semantics by means of SOS rules with data. Using this explicit specification, Section 3 presents a symbolic version of the SOS rules, together with the results that show its conformance to the original version. Section 4 sketches the implementation of a simulator based on the symbolic rules. The preservation of strong bisimulation, and the use of meta-theories is discussed in Section 5.

2 Syntax and Semantics of CIF

This section presents the syntax and semantics of a subset of CIF. First we introduce the syntactic elements of the language, and their formal definition. Then, the semantics of CIF is given in terms of SOS rules with data, and at the same time the problems described in the introduction are pointed out at each rule.

The basic building blocks of CIF are *automata*. They resemble the hybrid automata as presented in [10], which model computational and physical behavior of a system by mixing automata theory with the theory of differential algebraic equations.

Informally, a basic CIF automaton is shown in Figure 1, which models a conveyor belt carrying bottles to be filled. The volume of the bottle (V_B) cannot exceed the maximum allowed volume (V_{max}). The rate at which liquid enters the bottle is represented by variable Q. A clock c is used to ensure that Δ time units will elapse between the filling of two bottles.



Conveyor
cont control : V_B;
clock control : c;
sync : *open*, *close*;
$c = 0$
Moving
tcp : $c \leq 2 * \Delta$
inv : $\dot{c} = 1$
when $\Delta \leq c$ **act** *open* **do** $(\{V_B\}, {V_B}^+ = 0)$
Filling
tcp : true
inv : $V_B \leq V_{max} \wedge \dot{V_B} = Q$
when true **act** *close* **do** $(\{c\}, c^+ = 0)$

Fig. 1. Model of a conveyor (without operators).

The automaton shown above consists of two *locations*, *Moving* and *Filling*, which are depicted as circles. Locations represent the computational states of a system. Every location contains a predicate called *invariant*, which must hold as long as the system is in that state; and a *time can progress* (tcp) predicate, which must hold during time delays. For instance, location *Moving* has the predicate $\dot{c} = 1$ as invariant, and the predicate $c \leq 2 * \Delta$ as tcp.

The automaton of Figure 1 has two edges, which are depicted as arrows. Every edge contains a predicate called *guard* ($\Delta \leq c$ and true in Figure 1) that determines under which conditions a transition can be executed, a predicate called *update* ($V_B^+ = 0$ and $c^+ = 0$, where x^+ denotes the value of variable x in the next state) that determines how the model variables change after performing the action, and a set of *jumping variables* ($\{V_B\}$ and $\{c\}$) that specify the variables that are changed by the action. Edges are labeled by *actions* (*open* and *close* in Figure 1) that may be used to synchronize the behavior of automata in a parallel composition.

Every location has an *initialization predicate* associated to it, which describes constraints that the initial values of variables must satisfy if execution is to start in that location. Note that this predicate can be used to specify the initial locations of an automaton. In Figure 1, location *Moving* has $c = 0$ as its initial condition (depicted as an small incoming arrow without source location), and location *Filling* has the predicate false as initial condition (depicted by the absence of such an incoming arrow), which means that execution cannot begin in that computational state.

Every automaton declares a set of actions that must be synchronized if it is composed in parallel. In Figure 1 this set equals $\{open, close\}$, and it is specified by means of the **sync** keyword.

To model constraints in the joint evolution of a variable and its dotted version, the concept of *dynamic type* is used [15]. In Figure 1, variable c is associated to the **clock** keyword, which implies that during a time delay the value of c changes at rate 1. Variable V_B has the *continuous* dynamic type, declared using the **cont** keyword, which means that the value of variable V_B evolves as a continuous function of time, and variable $\dot{V_B}$ behaves as its derivative.

An automaton can contain *control variables* [7], which are variables that can be changed only by the automaton that declares them as such, and their values cannot jump arbitrarily when an action is performed. In Figure 1, the set of control variables is $\{V_B, c\}$, and graphically we use the **control** keyword in the variables declaration to specify such a set.

Formally, the locations of a CIF automata are taken from the set $\mathcal{L}$. Actions belong to the set $\mathcal{A}$. We use the symbol τ ($\tau \notin \mathcal{A}$) to refer to the silent action, and we define $\mathcal{A}_\tau \triangleq \mathcal{A} \cup \{\tau\}$. We distinguish the following types of variables: regular variables, denoted by the set $\mathcal{V}$; the dotted versions of those variables, which belong to the set $\dot{\mathcal{V}} \triangleq \{\dot{x} \mid x \in \mathcal{V}\}$; and *step variables*, which belong to the set $\{x^+ \mid x \in \mathcal{V} \cup \dot{\mathcal{V}}\}$. The values of the variables belong to the set Λ that contains the sets of booleans $\mathbb{B}$, and reals $\mathbb{R}$, among others. Guards are taken from the set $\mathcal{P}_g$; initialization predicates, invariants, and tcp predicates are taken from the sets $\mathcal{P}_t$; and update predicates are taken from the set $\mathcal{P}_r$. Expressions are taken from the set $\mathcal{E}$. Given an expression e and a value v, we assume $e = v$ to be an element of $\mathcal{P}_i$, $i \in \{g, t, r\}$; and we assume that the predicates are closed under conjunction.

Given these preliminaries, an automaton can be defined as follows:

Definition 1 (Automaton). *An automaton is a tuple*

$$(V, \text{init}, \text{inv}, \text{tcp}, E, \text{var}_C, \text{act}_S, \text{dtype})$$

where $V \subseteq \mathcal{L}$ is the set of locations; $\text{init}, \text{inv}, \text{tcp}\colon V \to \mathcal{P}_t$ *are functions that associate to each location its corresponding initialization predicate, invariant, and tcp predicate, respectively; $E \subseteq V \times \mathcal{P}_g \times \mathcal{A}_\tau \times (2^{\mathcal{V} \cup \dot{\mathcal{V}}} \times \mathcal{P}_r) \times V$ is the set of edges;* $\text{var}_C \subseteq \mathcal{V}$ *is the set of control variables;* $\text{act}_S \subseteq \mathcal{A}$ *is the set of synchronizing actions; and* $\text{dtype}\colon \mathcal{V} \rightharpoonup 2^{(\mathbb{T} \rightharpoonup \Lambda) \times (\mathbb{T} \rightharpoonup \Lambda)}$ *is the dynamic type mapping.*

In the previous definition, we used $A \rightharpoonup B$ is used to denote the set of partial functions from A to B, and $\mathbb{T}$ is the set of all time points.

Starting from an automaton, more complex models can be constructed by means of different operators. These include *parallel composition*, and the *urgency operator* to declare actions as urgent. The intuition behind an urgent action a is that time cannot pass if a is enabled. Section 2.1 presents the formal details.

We use the term *composition* to refer to a model that contains zero or more of these operators. The set $\mathcal{C}$ refers to the set of all compositions, and is formally defined as follows.

Definition 2 (Compositions). *The set of all compositions is defined by the following abstract grammar:*

$$\mathcal{C} ::= \alpha | \mathcal{C} \parallel \mathcal{C} | \upsilon_{a_\tau}(\mathcal{C})$$

where $a_\tau \in \mathcal{A}_\tau$.

2.1 Semantics of CIF

The semantics of CIF compositions is given in terms of SOS rules, which induce hybrid transition systems (HTS) [6]. The states of the HTS are of the form (p, σ), where $p \in \mathcal{C}$ is a composition, $\sigma \in \Sigma$ is a valuation, and $\Sigma \triangleq (\mathcal{V} \cup \dot{\mathcal{V}}) \to \Lambda$. There are three kind of transition in the HTS [1], namely, *action*, *time*, and *environment transitions*. We describe them in detail next.

Action transitions are of the form $(p, \sigma) \xrightarrow{a,b,X} (p', \sigma')$, and they 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, and label X is a set of control variables.

[1] Note that environment transitions are not present in the hybrid transition systems of [6].

Time behavior is captured by *time transitions*, which are of the form $(p, \sigma) \overset{\rho, A, \theta}{\longmapsto} (p', \sigma')$, and they model the passage of time in composition p, in an initial valuation σ, which results in a composition p' and valuation σ'. Function $\rho\colon \mathbb{T} \rightharpoonup \Sigma$ is the variable trajectory that models the evolution of variables during the time delay. Function $\theta : \mathbb{T} \rightharpoonup 2^{\mathcal{A}}$ is called *guard trajectory* [25], and it models the evolution of enabled actions during time delays. For each time point $s \in \mathrm{dom}(\theta)$, the function application $\theta(s)$ yields the set of enabled actions of composition p at time s. For every time transition ρ, $\mathrm{dom}(\rho) = [0, t]$, for some positive time point $t \in \mathbb{T}$, and $\mathrm{dom}(\rho) = \mathrm{dom}(\theta)$. Finally, label A contains the set of synchronizing actions of p and p' [2].

The last kind of transitions is *environment transitions*, which are of the form $(p, \sigma) \overset{A}{\dashrightarrow} (p', \sigma')$ and they model the fact the initial conditions and invariants of p (p' respectively) are satisfied in σ (σ'), and all the control variables of compositions p and p' are not changed in σ' (with regard to σ). Label A is the set of synchronizing actions of p and p'.

Next, we turn our attention to the SOS rules, which define the semantics of CIF compositions in terms of hybrid transition systems.

Rule 1 models the state change caused by the execution of an action. An action can be triggered in a state with valuation σ, only when there is an edge of the form $(v, g, a, (W, r), v')$, such that the initialization predicate $\mathrm{init}(v)$, the guard g, and the invariant of v are satisfied in σ; and it is possible to find a new valuation σ' such that the variables in the set $(X \cup \mathrm{var}_C) \setminus W$ do not change in σ', the update predicate r is satisfied in $\sigma'^{+} \cup \sigma$, and the invariant of the new location is satisfied in σ'. In Rule 1, id_v denotes the function that returns true only if the location equals v. More specifically, $\mathrm{id}_v \in V \to \mathbb{B}$, and $\mathrm{id}_v(w) \triangleq\ v \equiv w$, where $\equiv$ denotes syntactic equivalence. Given a function f, and a set A, $f \restriction_A$ is the restriction of f to A. The predicate $\sigma \models e$ denotes that e is satisfied (true) in σ. For a valuation σ we define $\sigma^{+} \triangleq \{(x^{+}, v) \mid (x, v) \in \sigma\}$.

$$\frac{\begin{array}{c}(v, g, a, (W, r), v') \in E, \sigma \models \mathrm{init}(v), \sigma \models g, \sigma \models \mathrm{inv}(v), \sigma' \models \mathrm{inv}(v'),\\ \sigma'^{+} \cup \sigma \models r, \sigma \restriction_{(X \cup \mathrm{var}_C) \setminus W} = \sigma' \restriction_{(X \cup \mathrm{var}_C) \setminus W}\end{array}}{\begin{array}{c}((V, \mathrm{init}, \mathrm{inv}, \mathrm{tcp}, E, \mathrm{var}_C, \mathrm{act}_S, \mathrm{dtype}), \sigma) \xrightarrow{a, a \in \mathrm{act}_S, X}\\ ((V, \mathrm{id}_{v'}, \mathrm{inv}, \mathrm{tcp}, E, \mathrm{var}_C, \mathrm{act}_S, \mathrm{dtype}), \sigma')\end{array}}\ 1$$

Clearly this rule is not suitable for implementation because it requires the explicit construction of the set of successor states, which can be infinite since there may be infinite valuations σ' that satisfy $\sigma'^{+} \cup \sigma \models r$. As we show later on, this situation becomes more complicated when we consider control variables and synchronizing actions in the context of parallel composition.

Rule 2 models the passage of time for an automaton. Time can pass if the invariant associated with the active locations is satisfied in each point in $[0, t]$, the time can progress predicate is satisfied in $[0, t)$, and the dynamic type constraints specified by dtype are satisfied. In this rule, given a variable x and a trajectory ρ, $\mathrm{dom}(\rho_x) = \mathrm{dom}(\rho)$ and $\rho_x(s) = \rho(s)(x)$ for all $s \in \mathrm{dom}(\rho)$.

[2] The set of synchronizing actions is not changed by transitions.

$$\frac{\begin{array}{c}\mathrm{dom}(\rho) = [0,t], 0 < t, \mathrm{dom}(\rho) = \mathrm{dom}(\theta), \rho(0) \models \mathrm{init}(v),\\ \langle \forall s : s \in \mathrm{dom}(\theta) : \theta(s) = \{a | (v,g,a,u,v') \in E \wedge \rho(s) \models g\}\rangle,\\ \langle \forall s : s \in [0,t] : \rho(s) \models \mathrm{inv}(v)\rangle, \langle \forall s : s \in [0,t) : \rho(s) \models \mathrm{tcp}(v)\rangle,\\ \langle \forall x : x \in \mathrm{dom}(\mathrm{dtype}) : (\rho_x, \rho_{\dot{x}}) \in \mathrm{dtype}(x)\rangle\end{array}}{\begin{array}{c}((V, \mathrm{init}, \mathrm{inv}, \mathrm{tcp}, E, \mathrm{var}_C, \mathrm{act}_S, \mathrm{dtype}), \rho(0)) \overset{\rho, \mathrm{act}_S, \theta}{\longmapsto}\\ ((V, \mathrm{id}_v, \mathrm{inv}, \mathrm{tcp}, E, \mathrm{var}_C, \mathrm{act}_S, \mathrm{dtype}), \rho(t))\end{array}} \; 2$$

Variable a guard trajectories (ρ and θ respectively) provide a convenient way of specifying the properties of a time delay. However, they are infinite objects, and this make them unsuitable for implementation purposes. Also notice that, in general, there may be an infinite number of outgoing time transitions (since there may be an infinite number of possible trajectories).

We consider next the rules for parallel composition. Rule 3 states that two synchronizing actions with the same label can execute in parallel only if they share the same initial and final valuation, and if the action is synchronizing in both compositions.

$$\frac{(p,\sigma) \xrightarrow{a,\mathrm{true},X} (p',\sigma'), (q,\sigma) \xrightarrow{a,\mathrm{true},X} (q',\sigma')}{(p \parallel q, \sigma) \xrightarrow{a,\mathrm{true},X} (p' \parallel q', \sigma')} \; 3$$

Rule 4 models interleaving behavior of two compositions when executed in parallel (for the sake of brevity we omit the symmetric rule). In these rules, an action can be performed in one of the components only if the initial and final valuations are consistent with the other composition, and if this action is not synchronizing in the other component, which is expressed by the condition $a \notin A$.

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

Rule 3 models in a succinct way the fact that the variables controlled by p (q respectively) are not changed by q (p): if p changes a variable which is not changed by q, then a transition involving q will not be possible. However, this simple specification complicates implementation since, according to it, determining whether a synchronizing action is possible amounts to finding a valuation σ' in the intersection of the sets:

$$\{\sigma' \mid \langle p,\sigma\rangle \xrightarrow{a} \langle p',\sigma'\rangle\} \text{ and } \{\sigma' \mid \langle q,\sigma\rangle \xrightarrow{a} \langle q',\sigma'\rangle\}$$

which cannot be computed in general. A similar phenomenon can be observed in Rule 4.

Rule 5 models the fact that if two compositions are put in parallel, time can pass t time units only if allowed by both partners. As can be seen in this rule, the set of enabled actions in the parallel composition at any point in time during the delay depends both on the set of enabled actions and the set of synchronizing actions in each component individually. In the expression $(\theta_p \cap \theta_q) \cup (\theta_p \setminus A_q) \cup$

$(\theta_q \setminus A_p)$ in the conclusion, the constants A_p and A_q, and the operators $\cap$ and $\setminus$ must be lifted accordingly to match the types. We avoid doing so here to keep the notation simple.

$$\frac{(p,\sigma) \overset{\rho, A_p, \theta_p}{\longmapsto} (p',\sigma'),\ (q,\sigma) \overset{\rho, A_q, \theta_q}{\longmapsto} (q',\sigma')}{(p \parallel q, \sigma) \overset{\rho, A_p \cup A_q, (\theta_p \cap \theta_q) \cup (\theta_p \setminus A_q) \cup (\theta_q \setminus A_p)}{\longmapsto} (p' \parallel q', \sigma')}\ 5$$

In the previous rule we can see that the time can progress condition of a composition is not straightforward to calculate since it depends not only on the tcp predicates of the individual automata, but also on the urgency conditions on the model, which are encoded in the guard trajectory. This trajectory is calculated using the set of enabled actions at a given time in the parallel composition $p \parallel q$: let A_i be the set of synchronizing actions of composition i, $i \in \{p, q\}$, and $\theta_i(s)$ the set of enabled actions of i at time s, then an action a is enabled at time s in $p \parallel q$ if at least one of the following conditions holds:

1. Action a is enabled in both components, which can be expressed as $a \in \theta_p(s) \cap \theta_q(s)$. A simple case analysis will convince the reader that when an action is enabled in both components, then it is enabled in the parallel composition of these, irrespective of whether a is synchronizing.
2. Action a is enabled in p, and is not synchronizing in q, which can be expressed as $a \in \theta_p(s) \setminus A_q$.
3. Symmetrically, $a \in \theta_q(s) \setminus A_p$.

Rule 5 exposes another problem with the explicit semantics of CIF. To obtain the sets of enabled actions during a delay, it is necessary to perform an infinite number of set operations on the trajectories. Rule 5 also states implicitly that the variable trajectories allowed in the conclusion is the intersection of the variable trajectories allowed by each component, and therefore it suffers the same problem as action transitions: to compute a variable trajectory, it is necessary to find a valuation in the intersection of the sets of trajectories allowed by p and the set of trajectories allowed by q.

Finally, we present the rule for the urgent action operator. Rule 6 specifies that the urgency operator restricts the time behavior of a composition in such a way that time can pass for as long as no urgent action is enabled. As it can be seen, although guard trajectories are not suitable for implementation purposes, they give a simple and intuitive semantics to urgency.

$$\frac{\mathrm{dom}(\rho) = [0,t], (p,\sigma) \overset{\rho, A, \theta}{\longmapsto} (p',\sigma'),\ \langle \forall s : s \in [0,t) : a \notin \theta(s) \rangle}{(\upsilon_a(p), \sigma) \overset{\rho, A, \theta}{\longmapsto} (\upsilon_a(p'), \sigma')}\ 6$$

3 Symbolic Semantics of CIF

Starting from the set of SOS rules given in Section 2.1, it is our goal to obtain a set of SOS rules without valuations, which constitutes a more concrete specification of the behavior of CIF compositions that is suitable for the implementation of an interpreter. To show that the symbolic rules correctly capture the explicit rules we establish correctness and completeness results.

The first problem that we pointed out is that the explicit rules with data induce infinitely branching transition systems. In Rule 1 we can see that the set of possible initial valuations of a transition are given by the guard, invariant, and initial condition associated to a given action. The final valuation is determined by the reset predicate. Thus, these predicates gives us a finite representation of the set of predecessor and successor states [9]. In the case of CIF we also need to know which variables are controlled and which variables can change after performing the action.

From the above observations, we conclude that we need symbolic action transitions of the form

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

where a is an action label; b is a boolean that determines whether action a is synchronizing; u (initial condition), g (guard), and n (invariant) are the predicates that have to be satisfied by the initial valuation in the explicit transition system; n' (invariant) and r (update predicate) are the predicates that have to be satisfied by the final valuation; C is the set of control variables; and W is the set of variables that can change in the new valuation.

The relationship between the explicit and symbolic action transitions is captured in the next theorems.

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

1. $\langle p \rangle \xrightarrow{a,b,g,u,n,n',W,C,r} \langle p' \rangle$
2. $\sigma \models g$, $\sigma \models u$, $\sigma \models n$, $\sigma' \models n'$, *and* $\sigma \cup \sigma'^{+} \models r$
3. $\sigma \restriction_{(C \cup X) \setminus W} = \sigma' \restriction_{(C \cup X) \setminus W}$

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

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

1. $\langle p \rangle \xrightarrow{a,b,g,u,n,n',W,C,r} \langle p' \rangle$
2. $\sigma \models g$, $\sigma \models u$, $\sigma \models n$, $\sigma' \models n'$, *and* $\sigma \cup \sigma'^{+} \models r$
3. $\sigma \restriction_{(C \cup X) \setminus W} = \sigma' \restriction_{(C \cup X) \setminus W}$

For time transitions we adopt a similar a approach to obtain a finite representation of the set of possible flows: we use invariants and tcp predicates. However this is not enough since flows also depend on the dynamic types, thus they must be also added to the labels of the symbolic time transitions. In the case of the urgent actions, a symbolic representation can be obtained by adding the guards associated to the actions. Nevertheless this is not sufficient for obtaining a sound and complete representation of the time transitions: we also need a set of urgent guards, which represent additional tcp conditions.

From the stated above, it follows that we need symbolic time transitions of the form

$$\langle p\rangle \overset{u,n,w,G,A,P,U}{\longmapsto} \langle p'\rangle$$

where u is the initialization predicate, n is the invariant that has to be satisfied by the time delay, w is the tcp predicate, G is a dynamic type mapping, A is the set of synchronizing actions, P is a set of pairs of guards and enabled actions, U is a set of urgent guards.

The relationship between the explicit and symbolic time transitions is captured in the next theorems.

Theorem 3 (Soundness of time transitions). *For all p, p', u, n, w, G, A, P, U, ρ, θ, and t we have that if the following conditions hold:*

1. $\langle p\rangle \overset{u,n,w,G,A,P,U}{\longmapsto} \langle p'\rangle$
2. $\mathrm{dom}(\rho) = \mathrm{dom}(\theta) = [0,t] \wedge 0 < t$, $\rho(0) \models u$, $\langle \forall s : s \in [0,t] : \rho(s) \models n\rangle$, $\langle \forall s : s \in [0,t) : \rho(s) \models w\rangle$, *and* $\langle \forall x : x \in \mathrm{dom}(G) : (\rho_x, \rho_{\dot{x}}) \in G(x)\rangle$
3. $\langle \forall s : s \in [0,t] : \theta(s) = \{a \mid (g,a) \in P \wedge \rho(s) \models g\}\rangle$, *and* $\langle \forall s : s \in [0,t) : \{g \mid g \in U \wedge \rho(s) \models g\} = \emptyset\rangle$

then there is a explicit time transition $(p, \rho(0)) \overset{\rho,A,\theta}{\longmapsto} (p', \rho(t))$.

Theorem 4 (Completeness of time transitions). *For all p, p', ρ, A, and t we have that if there is a explicit time transition $(p, \rho(0)) \overset{\rho,A,\theta}{\longmapsto} (p', \rho(t))$ then there exists u, n, w, G, P, and U such that the following conditions hold:*

1. $\langle p\rangle \overset{u,n,w,G,A,P,U}{\longmapsto} \langle p'\rangle$
2. $\rho(0) \models u$, $\langle \forall s : s \in [0,t] : \rho(s) \models n\rangle$, $\langle \forall s : s \in [0,t) : \rho(s) \models w\rangle$, $\langle \forall x : x \in \mathrm{dom}(G) : (\rho_x, \rho_{\dot{x}}) \in G(x)\rangle$, *and* $\langle \forall s : s \in [0,t] : \theta(s) = \{a \mid (g,a) \in P \wedge \rho(s) \models g\}\rangle$
3. $\langle \forall s : s \in [0,t) : \{g \mid g \in U \wedge \rho(s) \models g\} = \emptyset\rangle$

We present now the symbolic rules for CIF that illustrate better the the approach. The full set of rules can be found in [23].

Rules 7 and Rule 8 define the sets of symbolic action and time transitions, respectively. The labels can be easily derived by considering the soundness theorems. The rule for action generates as many transition as outgoing edges are there in the source location (v). The time rule generates two time transitions per location. Since there are no urgent actions at the automaton level, the set of urgent actions on the arrow is defined to be empty.

$$\frac{(v, g, a, (W, r), v') \in E}{\begin{array}{c}\langle (V, \mathrm{init}, \mathrm{inv}, \mathrm{tcp}, E, \mathrm{var}_C, \mathrm{act}_S, \mathrm{dtype})\rangle \xrightarrow{a, a \in \mathrm{act}_S, g, \mathrm{init}(v), \mathrm{inv}(v), \mathrm{inv}(v'), W, \mathrm{var}_C, r} \\ \langle (V, \mathrm{id}_{v'}, \mathrm{inv}, \mathrm{tcp}, E, \mathrm{var}_C, \mathrm{act}_S, \mathrm{dtype})\rangle\end{array}} \quad 7$$

$$\frac{v \in V}{\langle (V, \mathrm{init}, \mathrm{inv}, \mathrm{tcp}, E, \mathrm{var}_C, \mathrm{act}_S, \mathrm{dtype})\rangle \xmapsto{\mathrm{init}(v), \mathrm{inv}(v), \mathrm{tcp}(v), \mathrm{dtype}, \mathrm{act}_S, \{(g,a) \mid (v,g,a,u,v') \in E\}, \emptyset} \langle (V, \mathrm{id}_v, \mathrm{inv}, \mathrm{tcp}, E, \mathrm{var}_C, \mathrm{act}_S, \mathrm{dtype})\rangle} \; 8$$

The rules for parallel composition, as opposed to the previous symbolic rules, are more complex. Consider Rule 9, the symbolic counterpart of the rule for synchronizing action behavior. It states that the set of variables that can be changed by the parallel composition is the intersection of the variables that can be changed by the two components. This, though it is the desired behavior, is not obvious from the explicit specification (Rule 3). The same observation can be made for the set of control variables associated to the action.

$$\frac{\langle p \rangle \xrightarrow{a, \mathrm{true}, g_p, u_p, n_p, n'_p, W_p, C_p, r_p} \langle p' \rangle, \langle q \rangle \xrightarrow{a, \mathrm{true}, g_q, u_q, n_q, n'_q, W_q, C_q, r_q} \langle q' \rangle}{\langle p \parallel q \rangle \xrightarrow{a, \mathrm{true}, g_p \wedge g_q, u_p \wedge u_q, n_p \wedge n_q, n'_p \wedge n'_q, W_p \cap W_q, (C_p \setminus W_p) \cup (C_q \setminus W_q), r_p \wedge r_q} \langle p' \parallel q' \rangle} \; 9$$

For defining the rule for time transitions, we make use of the operators $\Cap$, $\cap_2$, and $\setminus_2$, which are defined next.

Definition 3. *Given two partial functions* $f : A \rightharpoonup 2^B$ *and* $g : A \rightharpoonup 2^B$*,* $f \Cap g$ *is the function such that* $\mathrm{dom}(f \Cap g) = \mathrm{dom}(f) \cup \mathrm{dom}(g)$ *and for all* $x \in \mathrm{dom}(f \Cap g)$*:*

$$(f \Cap g)(x) = \begin{cases} f(x) \cap g(x) & \textit{if } x \in \mathrm{dom}(f) \cap \mathrm{dom}(g) \\ f(x) & \textit{if } x \in \mathrm{dom}(f) \setminus \mathrm{dom}(g) \\ g(x) & \textit{if } x \in \mathrm{dom}(g) \setminus \mathrm{dom}(f) \end{cases}$$

Given two sets $A, B \subseteq C \times D$*:*

$$A \cap_2 B \triangleq \{(c_a \wedge c_b, d) \mid (c_a, d) \in A \wedge (c_b, d) \in B\}$$

Given a set $A \subseteq B \times C$ *and a set* $D \subseteq C$*:*

$$A \setminus_2 D \triangleq \{(a, b) \mid (a, b) \in A \wedge b \notin D\}$$

Using the operators previously defined we can state the symbolic rule for time transitions in parallel compositions.

$$\frac{\langle p \rangle \xmapsto{u_p, n_p, w_p, G_p, A_p, P, U_p} \langle p' \rangle, \langle q \rangle \xmapsto{u_q, n_q, w_q, G_q, A_q, Q, U_q} \langle q' \rangle}{\langle p \parallel q \rangle \xmapsto{u_p \wedge u_q, n_p \wedge n_q, w_p \wedge w_q, G_p \Cap G_q, A_p \cup A_q, (P \cap_2 Q) \cup (P \setminus_2 A_q) \cup (Q \setminus_2 A_p), U_p \cup U_q} \langle p' \parallel q' \rangle} \; 10$$

The previous rule expresses that the dynamic typing constraints of the two components are intersected, as can be also inferred from the original semantics, but it is nevertheless implicit in the trajectories. Once the appropriate operators are defined, the symbolic guard trajectory is calculated in a similar way as its explicit version. The urgency conditions of both components are combined in the conclusion, which also corresponds with the intuition, but it is not evident from the explicit SOS rules.

The rule for urgency shows how the symbolic guard trajectory is used to obtain the urgency conditions associated to a given composition.

$$\frac{\langle p \rangle \overset{u,n,w,G,A,P,U}{\longmapsto} \langle p' \rangle}{\langle \upsilon_a(p) \rangle \overset{u,n,w,G,A,P,U \cup \{g \mid (g,a) \in P\}}{\longmapsto} \langle \upsilon_a(p') \rangle} \; 11$$

4 Using SOS for Implementing a Simulator

In this section we describe briefly how to implement a simulator for CIF, the hybrid language under consideration, using the symbolic rules. Aspects of the simulator such as equation solving can be handled in standard ways [14], and their discussion is outside the scope of this work. The full implementation of the simulator that uses the symbolic semantics of CIF can be found in [22].

The CIF simulator is a procedure that takes a pair (process) (p, σ) containing a CIF composition and a valuation, and it computes a random path in the hybrid transition system induced by the state (p, σ). An essential piece in the construction of such procedure are the *successor functions* [16], which contain all the information necessary to calculate the next step in a simulation.

In CIF there are two successor functions needed for the simulator: a *action successor function* $\mathcal{S}_a$, which given a composition p returns the set of all the symbolic outgoing action transitions from p; and a *time successor function* $\mathcal{S}_t$, which given a composition p returns the set of all symbolic outgoing time transitions from p. Consistency transitions are of interest only to compute the action functions and are not visible at the simulator level. As it turns out, these functions can be computed. This is a consequence of the following theorem, and the fact that the transition system induced by the symbolic rules is finitely branching.

Theorem 5 (Well-definedness of $\mathcal{S}_a$ and $\mathcal{S}_t$). *For each CIF composition $p \in \mathcal{C}$ it is possible to define functions $\mathcal{S}_a$ and $\mathcal{S}_t$ such that:*

$$\mathcal{S}_a(p) = \{(a, b, g, u, n, n', W, C, r, p') \mid \langle p \rangle \xrightarrow{a,b,g,u,n,n',W,C,r} \langle p' \rangle\}$$
$$\mathcal{S}_t(p) = \{(u, n, w, G, A, P, U, p') \mid \langle p \rangle \overset{u,n,w,G,A,P,U,p'}{\longmapsto} \langle p' \rangle\}$$

The definition of such functions is straightforward, and it can be obtained from the symbolic SOS rules.

Note that if we consider a hybrid language which contains recursion and parallel composition, then it is not longer the case that the transition system is always finitely branching [4]. However for guarded recursion we are guaranteed to obtain finitely branching processes.

Using these results, the problem of computing a successor state of a composition p in an initial valuation σ can be solved with the help of the functions $\mathcal{S}_a$ and $\mathcal{S}_t$. To find a successor state (p', σ') we need to consider the symbolic transitions in:

$$\mathcal{S}_a(p) \cup \mathcal{S}_t(p)$$

Then we pick one transition at a time, until we find a transition t and a valuation σ' such that:

– t is of the form $(a, b, g, u, n, n', W, C, r, p')$, σ satisfies $g \wedge u \wedge n$, and σ' is obtained by solving the equation:

$$n'^{+} \wedge r \wedge \bigwedge_{x \in C \setminus W} x = x^{+}$$

where all the free variables in $\mathrm{dom}(\sigma)$ are replaced by their actual values (given by σ itself).

– t is of the form $(u, n, w, G, A, P, U, p')$, σ satisfies $u \wedge n$, σ' is the evaluation of a trajectory at time t, where the trajectory is computed (numerically or symbolically) from the invariant n, and the time can progress condition

$$\neg w \wedge \bigwedge_{g \in U} \neg g$$

is used to determine the maximum duration of the delay.

If such transition cannot be found using the completeness result we can safely conclude that the system has deadlock.

5 Equational Theories via Symbolic SOS

In this section, we examine to what extent the symbolic SOS is useful for inferring properties about the explicit SOS. We consider the "preservation" of strong bisimilarity, and the potential use of the meta-theory in SOS for establishing congruence and properties of operators.

First we define the (standard) notions of strong bisimilarity[3] for SOS and for SSOS.

Definition 4 (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. $(p, \sigma) \overset{\ell}{\longmapsto} (p', \sigma') \Rightarrow \langle \exists q' :: (q, \sigma) \overset{\ell}{\longmapsto} (q', \sigma') \wedge (p', q') \in R \rangle$
3. $(p, \sigma) \overset{\ell}{\dashrightarrow} (p', \sigma') \Rightarrow \langle \exists q' :: (q, \sigma) \overset{\ell}{\dashrightarrow} (q', \sigma') \wedge (p', q') \in R \rangle$

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

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

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

[3] Note that the notion of strong bisimilarity coincides with that of *stateless bisimilarity*, given in [18].

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

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.

Strong bisimilarity is preserved by the symbolic SOS, which means that any equivalence valid in the symbolic SOS is also valid in the explicit version.

Theorem 6 (Preservation of Strong Bisimilarity). *For all closed terms p and q, $SSOS \models p \leftrightarrow q$ implies $SOS \models p \leftrightarrow q$.*

Proof. Using the soundness and completeness theorems it can be shown that any relation R that witnesses bisimilarity for SSOS is also a bisimulation relation that witnesses bisimilarity on SOS. □

It is easy to see that the converse of Theorem 6 does not hold. This is because symbolic SOS may have symbolic transitions that do not reflect any explicit transition (symbolic transitions do not take into account the satisfiability of predicates). Using the previous observation, one can see that it is not possible to prove in general that congruence is preserved by the symbolic SOS.

On the bright side, congruence formats exists that make it possible to prove congruence for SOS with data [18]. As it turns out, the explicit rules of CIF [3] conform to the *process-tyft format*, and therefore strong bisimilarity (stateless bisimilarity), is a congruence for all the operators.

As a consequence of removing the state part from such SOS, and Theorem 6 much of the meta-theory of SOS has become applicable: any axiom that is proven sound on the symbolic SOS is also sound in the explicit SOS.

In literature meta-theory is available for establishing axiomatic properties of operators based on syntax of deduction rules such as commutativity [17], associativity [5], idempotence [1], and unit and zero elements [2]. In the present setting these rule formats provides us with proof that parallel composition is commutative and associative.

Proposition 1. *The following properties hold:*

Commutativity $SSOS \models p \parallel q \;\leftrightarrow\; q \parallel p$
Associativity $SSOS \models (p \parallel q) \parallel r \;\leftrightarrow\; p \parallel (q \parallel r)$

Proof. Application of the comm-tyft format for the commutativity and associativity of parallel composition requires interpreting the labels as (arbitrary) representatives of equivalence classes.

Then since the symbolic deduction rules of CIF are in the comm-tyft format of [17, Definition 11] by [17, Theorem 12] commutativity follows.

The deduction rules for alternative composition and parallel composition are in the ASSOC-De Simone format of [5, Definition 8] and associativity of alternative and parallel composition thus follow from [5, Theorem 1]. □

6 Concluding Remarks

We have seen that the specification by means of structured operational semantics with data, while useful for specifying the behavior of (hybrid) languages in a succinct way, is not appropriate for guiding the implementation of interpreters for these languages.

To derive a simulator from this abstract specification, we have made use of the symbolic SOS approach, obtaining a set of SOS rules without data, which are proven to be a sound and complete representation of the original SOS, and at the same time they serve as the basis of the simulator. Note that even though we have illustrated our approach using CIF, it can be used for other hybrid languages.

We plan to exploit the fact that, in the case of CIF, the transition system induced by the symbolic rules is finite, to obtain a linearization algorithm in the spirit of [13,24]. These finite transition systems could be also used for model transformations between CIF and other languages.

Currently we have not found a way of dealing with the variable scope operator as it is specified in [3]. The problem is that the semantic rule for this operator mixes syntactic and semantic information. We have to investigate if a solution can be found by means of an alternative formulation of the rule.

Another interesting line of future work is to analyze the possibility of obtaining symbolic model checking algorithms from symbolic SOS specifications.

References

1. L. Aceto, A. Birgisson, A. Ingolfsdottir, M. Mousavi, and M. A. Reniers. Rule formats for determinism and idempotency. In *Proceedings of the 3rd International Conference on Fundamentals of Software Engineering (FSEN'09)*, Lecture Notes in Computer Science, Kish Island, Iran, 2009. Springer-Verlag, Berlin, Germany.
2. L. Aceto, M. Cimini, A. Ingólfsdóttir, M. R. Mousavi, and M. A. Reniers. On rule formats for zero and unit elements. *Electr. Notes Theor. Comput. Sci.*, 265:145–160, 2010.
3. J. Baeten, D. van Beek, D. Hendriks, A. Hofkamp, D. N. Agut, J. Rooda, and R. Schiffelers. Definition of the compositional interchange format. Technical Report Deliverable D1.1.2, Multiform, 2010.
4. J. C. M. Baeten, T. Basten, and M. A. Reniers. *Process Algebra: Equational Theories of Communicating Processes (Cambridge Tracts in Theoretical Computer Science).* Cambridge University Press, 1 edition, December 2009.
5. S. Cranen, M. Mousavi, and M. A. Reniers. A rule format for associativity. In F. van Breugel and M. Chechik, editors, *Proceedings of the 19th International Conference on Concurrency Theory (CONCUR'08)*, volume 5201 of *Lecture Notes in Computer Science*, pages 447–461, Toronto,Canada, 2008. Springer-Verlag, Berlin, Germany.
6. P. Cuijpers, M. Reniers, and W. Heemels. Hybrid transition systems. Technical Report CS-Report 02-12, TU/e, Eindhoven, Netherlands, 2002.
7. G. Frehse. *Compositional Verification of Hybrid Systems using Simulation Relations.* PhD thesis, Universiteit of Nijmegen, 2005.

8. G. Frehse. PHAVer: Algorithmic verification of hybrid systems past HyTech. In M. Morari and L. Thiele, editors, *Hybrid Systems: Computation and Control, 8th International Workshop*, volume 3414 of *Lecture Notes in Computer Science*, pages 258–273. Springer-Verlag, 2005.
9. M. Hennessy and H. Lin. Symbolic bisimulations. In *Selected papers of the meeting on Mathematical foundations of programming semantics*, pages 353–389, Amsterdam, The Netherlands, The Netherlands, 1995. Elsevier Science Publishers B. V.
10. T. A. Henzinger. The theory of hybrid automata. In M. K. Inan and R. P. Kurshan, editors, *Verification of Digital and Hybrid Systems*, volume 170 of *NATO ASI Series F: Computer and Systems Science*, pages 265–292. Springer-Verlag, New York, 2000.
11. H. Highly-complex and networked control systems. http://www.hycon2.eu/, 2010.
12. HYCON Network of Excellence. http://www.ist-hycon.org/, 2005.
13. U. Khadim, D. A. v. Beek, and P. J. L. Cuijpers. Linearization of hybrid Chi using program counters. Technical Report CS-Report 07-18, Eindhoven University of Technology, Department of Computer Science, The Netherlands, 2007.
14. E. A. Lee and H. Zheng. Operational semantics of hybrid systems. In *Hybrid Systems: Computation and Control (HSCC), volume LNCS 3414*, pages 25–53. Springer-Verlag, 2005.
15. N. Lynch, R. Segala, and F. Vaandrager. Hybrid I/O automata revisited. In *Proceedings Fourth International Workshop on Hybrid Systems: Computation and Control (HSCC'01)*, pages 403–417. Springer-Verlag, 2001.
16. K. L. Man and R. R. H. Schiffelers. *Formal Specification and Analysis of Hybrid Systems*. PhD thesis, Eindhoven University of Technology, 2006.
17. M. Mousavi, M. Reniers, and J. F. Groote. A syntactic commutativity format for SOS. *Information Processing Letters (IPL)*, 93:217–223, Mar. 2005.
18. M. R. Mousavi, M. A. Reniers, and J. F. Groote. Notions of bisimulation and congruence formats for SOS with data. *Information and Computation*, 200(1):107–147, 2005.
19. MULTIFORM consortium. Integrated multi-formalism tool support for the design of networked embedded control systems MULTIFORM. http://www.multiform.bci.tu-dortmund.de, 2008.
20. D. Nadales Agut, M. A. Reniers, R. Schiffelers, K. Jørgensen, and D. A. v. Beek. A semantic-preserving transformation from the compositional interchange format to uppaal. In *18th Triennial World Congress of the International Federation of Automatic Control*, Milano, 2011. CD-ROM.
21. G. D. Plotkin. A structural approach to operational semantics. *Journal of Logic and Algebraic Programming*, 60-61:17–139, 2004.
22. Systems Engineering Group TU/e. CIF toolset. http://devel.se.wtb.tue.nl/trac/cif, 2011.
23. Systems Engineering Group TU/e. Publications on CIF 2. http://se.wtb.tue.nl/sewiki/cif/publications, 2011.
24. Y. S. Usenko. *Linearization in μCRL*. PhD thesis, Eindhoven University of Technology, 2002.
25. D. van Beek, P. Cuijpers, J. Markovski, D. Nadales Agut, and J. Rooda. Reconciling urgency and variable abstraction in a hybrid compositional setting. In K. Chatterjee and T. Henzinger, editors, *Formal Modeling and Analysis of Timed Systems*, volume 6246 of *Lecture Notes in Computer Science*, pages 47–61. Springer Berlin / Heidelberg, 2010.