Reconciling Urgency and Variable Abstraction in a Hybrid Compositional Setting

D.A. van Beek, P.J.L. Cuijpers, J. Markovski, D.E. Nadales Agut, and J.E. Rooda [⋆]

Eindhoven University of Technology (TU/e)
P.O. Box 513, NL-5600 MB Eindhoven, The Netherlands.

Abstract. The extension of timed formalisms to a hybrid setting with urgency, has been carried out in a rather straightforward manner, seemingly without difficulty. However, in this paper, we show that the combination of urgency with abstraction from continuous variables leads to undesired behavior. Abstraction from continuous variables ultimately leads to a timed system again, but with a much richer set of possible branching behaviors than a timed system that comprises only clocks. As it turns out, the formal definition of urgency, as defined for timed systems with clocks, does not fit our intuition of urgency anymore when applied to a timed system that is an abstraction of a hybrid system. Therefore, we propose to extend the formal semantics of timed and hybrid systems with guard trajectories. In this way, the continuous branching behavior introduced by hybrid systems remains visible even after abstraction from continuous variables. The practical applicability of the introduction of guard trajectories is illustrated by our revision of the structured operational semantics of the CIF language. The interplay between urgency and abstraction now adheres to our intuition, while compositionality with respect to urgency, variable abstraction, and parallel composition, is retained. In the future, symbolic elimination of urgency can be used to ensure that guard trajectories do not need to be actually calculated.

1 Introduction

Urgent actions were introduced in timed formalisms to support easy modeling of greedy, or eager, behavior. As an example, we consider timed automata of [1]. These timed automata are extensions of standard automata with clocks that keep track of passage of time. The actions of an automaton are guarded by constraints on the clocks, which indicate when the action *may* be performed. In addition, the introduction of urgency of actions to this model allows the modeler to determine when an action *must* be performed. Using model checkers like UPPAAL, KRONOS, and IF [2–4], one can then check whether the urgent execution of actions will guarantee that these actions meet their deadlines.

[⋆] This work was partly done as part of the EU FP7 2007-2013 projects MULTIFORM and C4C, contract numbers FP7-ICT-224249 and FP7-ICT-223844

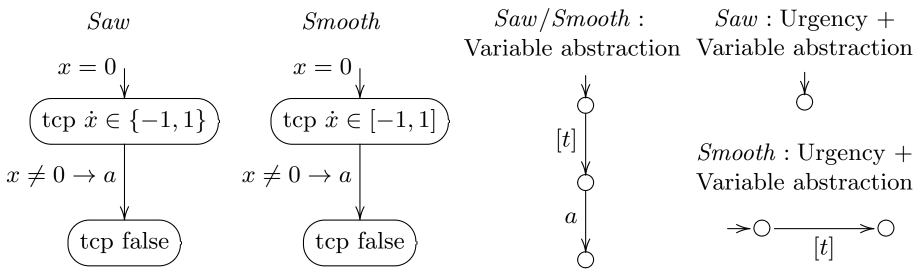
The extension of these timed formalisms to a hybrid setting with urgency has been carried out in a rather straightforward manner. In hybrid automata [5, 6] clocks are generalized to differential equations over continuous variables. Thus, guards deal with continuous variables, so urgency allows the modeler to specify that an action will happen *as soon as* a guard becomes true. Tools like HyTech and HyVisual [7, 8] already support this idea, e.g., they can be used to verify that an action is executed before a certain continuous condition is met.

Admittedly, the formal definition of urgency has had its complications. One problem was that the earliest possible moment of an action may not be defined, e.g., when the guard on an action is a left-open time-interval. Another problem was that the unexpected blocking of urgent actions due to a failing synchronization may enable other actions that were not available before. In this way, systems that were considered equivalent before placing them in a parallel composition, may behave differently after placing them in a parallel composition, thus ruining compositionality. Still, most of these complications have been solved for timed systems. Thus, one often defines that the urgent execution of an action in a left-open time-interval leads to a deadlock, and one can strengthen the notion of equivalence to consider non-urgent actions, even if there are earlier urgent actions that prevent them. These solutions, even though not the only possible ones, are satisfactory and readily transferrable to a hybrid setting as well.

An additional complication in the definition of urgency arises only when one starts in a hybrid setting and subsequently obtains a timed system through abstraction of hybrid variables. In fact, the problem already manifests itself when only part of the continuous behavior is abstracted from, but for the sake of simplicity we abstract from all variables in the setting of this paper. To the best of our knowledge, the combination of urgency and variable abstraction has not been studied in detail before, hence the problem did not manifest itself sooner.

When abstracting from the value of continuous variables in a hybrid system, one would like to obtain a timed system in which the moments at which certain urgent actions are enabled or disabled are accurately preserved. The fact that the value of a continuous variable is not directly observable, should not change the fact that a certain guard that depends on this value is true or false at a certain time. Furthermore, we would expect in particular that if a system contains a deadlock *before* abstracting from the value of continuous variables, that it also contains this deadlock *after* abstracting from the value of these variables. After all, the abstraction is intended to give us a system that is easier to analyse, which means that the abstraction should preserve the properties that we are interested in. Finally, we expect that abstraction distributes over compositions and operators whenever reasonably possible, because we would like to employ abstraction to verify separate components. In the opposite case, the results of such a partial verification could not be used in the verification of the whole.

The definitions for abstraction from variables in current literature, e.g. [9, 10], aim towards obtaining a timed transition system when all variables are abstracted from. However, in a timed transition system the continuity of the moments of choice is abstracted from, which becomes even more prominent when urgency is introduced. We note that other typical compositions and operators in the automata- and process-theoretic approaches are not affected. As an illustration of how urgency depends on the continuity of the moments of choice, let us consider the hybrid systems depicted in Fig. 1. The systems differ only in the differential inclusions that define their continuous behavior. *Saw* has a differential inclusion ranging over two points, while *Smooth* has a differential inclusion ranging over the closed interval between these two points.



Saw
$x = 0$
tcp $\dot{x} \in \{-1, 1\}$
$x \neq 0 \to a$
tcp false
Smooth
$x = 0$
tcp $\dot{x} \in [-1, 1]$
$x \neq 0 \to a$
tcp false
Saw/*Smooth* : Variable abstraction
$[t]$
a
Saw : Urgency + Variable abstraction
Smooth : Urgency + Variable abstraction
$[t]$

Fig. 1. *μCIF* automata *Saw* and *Smooth*, and the resulting infinite timed transitions systems parameterized with $t \in \mathbb{R}$ such that $t > 0$ after applying: (1) variable abstraction and (2) urgency followed by variable abstraction

Using the existing definitions of urgency, we can show that any behavior of *Saw* can be mimicked by *Smooth*, but not vice versa. In particular, from the initial valuation $x \mapsto 0$, *Smooth* allows x to remain 0 for an arbitrary period of time (thus disallowing the action a), while *Saw* does not allow x to remain 0 and it is always able to execute an a after an arbitrarily small delay. Declaring the action a to be urgent, now shows the difference between *Saw* and *Smooth* even more prominently, because the action a must happen as soon as it can. The application of urgency to *Saw* leads to a deadlock, since the guard leads to a left-open time-interval in which a is enabled. The latter gives us a system in which $x(t) = 0$ is the only possible solution, so a will never occur, but time can progress.

The above described behavior supports the intuition, but when we abstract from the value of x before applying urgency, the semantics changes unexpectedly. The abstraction from x results in a isomorphic timed transition system for both *Saw* and *Smooth*, as each variable trajectory in *Smooth* has a related trajectory in *Saw* with the same duration and the same start- and end-valuation. Making a urgent leads to a system in which no transition labeled by a can occur, but time can progress. Apparently, by abstracting from x, we also abstracted from the intervals in which a is enabled, causing the deadlock to disappear. This is not desirable, since it allows for variable abstraction only on the 'top' level, after the system has been described completely and urgency has been applied. Therefore, we cannot employ variable abstraction to simplify the verification of the components of a system.

As a remedy, we propose to make the guard trajectories visible on the timed transitions, even if one has already abstracted from the values of the variables that these guards comprise. To illustrate our approach, we adapt the structured operation semantics (SOS) [11] of a subset of the compositional interchange format (*CIF*) language [12, 13] that comprises urgency and variable abstraction. Another solution to the above problem would be to restrict to hybrid systems in which the above phenomenon does not occur. These systems, referred to as *finite set refutable*, were studied in [14]. All timed systems that comprise fixed clock-rates are finite-set refutable, as well as all hybrid systems that employ only differential equations, rather than differential inclusions. However, open systems with free input/output variables and systems with open differential inclusions are, in general, not finite-set refutable. So, while the extension to guard trajectories does not change anything for classical timed system, it provides a more applicable theory with a more robust notion of variable abstraction. Furthermore, ongoing research indicates that symbolic elimination of the urgency operator is possible, meaning that the introduced guard trajectories never need to be actually calculated when analyzing the behavior of a system.

The remainder of this paper is structured as follows. We discuss related work in section 2. Section 3 illustrates our approach by adapting the SOS of a relevant subset of the *CIF* language. We show that urgency and variable abstraction indeed distribute modulo bisimulation, solving the aforementioned problems. In section 4 we show compatibility with the common urgency concepts from the literature, and we end with concluding remarks in section 5.

2 Related work

In this section we discuss the notions of urgency in the literature on timed and hybrid automata and relate them to our work.

Specifying urgency Early approaches to specifying urgency employed state invariants to limit the progress of time at a given location [15]. Amongst others, this approach found its way into the toolset UPPAAL [2]. A variant of this approach, termed stopping conditions, has been employed in a timed restriction of the hybrid I/O automata [16]. When the stopping condition becomes valid, the progress of time is stopped and an activity must execute immediately. Another extension, termed timed automata with deadlines, was investigated in [17–20]. Deadlines are auxiliary clock constraints, which specify when the transition has become urgent. It is required that they imply the corresponding guard constraints to ensure that at least one transition has been enabled after stopping the progress of time. This property of progress of time is also known as time reactivity or time-lock freedom [19, 21]. Timed automata with deadlines are embedded in the specification languages IF and MODEST [4, 22]. Yet another approach employing urgency predicates has been proposed in [23] as an extension of timed I/O automata. In the same study the four approaches from above have been paralleled, concluding that stopping conditions, deadlines, and urgency predicates essentially have the same expressivity, whereas invariants can be additionally applied to right-open intervals as well. However, by construction only deadlines and urgency predicates guarantee time reactivity.

In the hybrid setting, urgency flags denote urgent actions in the framework of HyTech [7] and invariants are required to endorse urgent transitions. The same approach is used when coupling Hytech and UPPAAL [24]. The completely opposite approach is taken in HyVisual [8], where every action considered urgent.

In this paper, we consider two types of urgency: (1) global labeled transition urgency by means of urgent action labels and (2) local urgency by means of the time-can-progress construct. The former is implemented by means of an urgency mapping in a fashion similar to the urgency flags of [7], whereas as the latter is closest to the stopping conditions of [16]. As a design choice, we do not hard-code/enforce the property of time reactivity in the semantics, although we foresee that it can be supported by restricting the allowed syntax.

Synchronization of urgent actions When considering synchronization of urgent actions, the literature provides three prominent manners: (1) synchronization with hard deadlines or impatient synchronization or AND-synchronization, where the urgency constraints of all synchronizing parties must be endorsed as soon as they are enabled, (2) synchronization with soft deadlines or patient synchronization or MAX-synchronization, where some urgency constraints can be contravened so that the synchronization can occur as long as the guards still hold, and (3) MIN-synchronization, which occurs when one of the synchronizing parties is ready to synchronize provided that the other will eventually be enabled. In the setting of this paper, we opt for both patient and impatient synchronization. Implementing MIN-synchronization would require substantial changes of the semantics, i.e., time look-ahead capability, whilst its practical use for modeling real-life systems is limited [25].

When using urgency flags, there are substantial restrictions on the guards of the synchronizing actions [7, 24]. In the current setting, we use an operator to specify synchronizing actions and urgency. The patient synchronization contravenes urgency constraints by assuming non-urgency of all synchronizing actions and, only after successful synchronization in accordance with the guards has succeeded, it re-imposes the urgency constraints. Our approach is similar to the drop bisimulation proposed in [20], where deadlines are dropped to enable patient synchronization and, afterwards, undropped to preserve compositionality. To support modeling with data we also introduced urgent channels in the extended framework of χ 2.0 [26], a topic beyond the scope of this paper.

3 μ*CIF*

This section presents a concise syntax and formal semantics of a subset of the *CIF* language [12], denoted as μ*CIF*. It illustrates how the urgency concept can be defined in a compositional way, robust against abstraction from local variables, by extending hybrid transition systems (HTSs) [27] with guard trajectories. Similar ideas are applied in the complete framework of χ 2.0 [26], and in the latest extension of CIF [28, 12]. *μCIF* is a modeling language for hybrid systems that adopts concepts from hybrid automata theory and process algebra. We briefly overview the features of the language.

The basic building blocks of *μCIF* are atomic automata, which consist of a set of locations $\mathcal{L}$ and edges that connect them. A location specifies a state of the system, and it can have equations associated to it in the form of *time can progress* predicates. These equations define the continuous behavior of the automaton: they determine when time can pass, and how the value of the variables changes as time elapses. The values of the variables belong to the set Λ that contains, among else, the sets $\mathbb{B}$, $\mathbb{R}$, and $\mathbb{C}$. Guarded labeled transitions specify the discrete behavior of the system. These transitions are labeled with actions originating from $\mathcal{A}$. Guards are incorporated as predicates over variables, given by $\mathcal{P}$. Continuous behavior is specified by timed transitions labeled by variable and guard trajectories as described below. The set $\mathcal{T}$ comprises all time points.

We distinguish between two types of variables: (1) variables, denoted by the set $\mathcal{V}$, and (2) the dotted versions of those variables, which belong to the set $\dot{\mathcal{V}} \triangleq \{\dot{x} \mid x \in \mathcal{V}\}$. The evolution of the value of a variable as time elapses is constrained by equations. Furthermore, we distinguish between *discrete variables*, which dotted versions are always 0, and *continuous variables*, which dotted versions represent the derivative. Variables are constrained by differential algebraic equations and we implement them as predicates, given by $\mathcal{P}$, over all variables $\mathcal{V} \cup \dot{\mathcal{V}}$. We also have initialization conditions, elements of $\mathcal{P}$, which allows to model steady state initialization.

We introduce several operators of *μCIF*. Parallel composition with synchronization composes two *μCIF* automata in parallel, synchronizing on a specified set of actions, while interleaving the rest. Actions are not synchronizing by default and they must be declared as such by placing them in the synchronization set. We introduce local urgency by means of a *time-can-progress* predicate and global urgency by means of *urgency composition.* The time-can-progress predicate is associated to each location of the automaton and specifies whether passage of time is allowed. Action transitions become urgent when time can no longer progress. The urgency operator declares action transitions as urgent, thus disabling the passage of time whenever an urgent action is enabled. Finally, variables in *μCIF* can be made local by means of *variable scopes.* When a variable is abstracted from, the changes made to the variable by the automaton are not visible outside the scope, and vice versa, external automata cannot change the value of the abstracted variable.

The semantics of the *μCIF* is given in the form of a structured operational semantics (SOS) [11] on a hybrid transition system(HTS) [27]. This is significantly different with the typical way of giving semantics to hybrid automata [5]. In the SOS approach, the semantics of the operators is defined by rules that give the semantics of the composition on the basis of the semantics of the constituent components, whereas in traditional approaches, the semantics of the composition is given by syntactic transformations of the constituent components to a basic automaton. In [28, 12], the SOS approach was chosen since it is better suited for guiding the implementation process [29]. Additionally, it makes use of standard SOS formats to show compositionality [30], thus enabling symbolic reasoning. Since *μCIF* is a subset of CIF, we inherit the SOS approach.

Syntax The basic building block of *μCIF* are atomic automata. An atomic automaton resembles a hybrid automaton: for each location in an automaton there is a predicate 'init' that specifies the conditions under which execution can begin on that location, and a predicate 'tcp' that specifies local urgency by giving the conditions under which time can progress in that location and that determines how the value of variables change over time, defining the dynamic behavior. The edges of the automaton describe the actions that the automaton can perform.

Definition 1. *An atomic automaton is a tuple* $(L, \mathrm{init}, \mathrm{tcp}, E)$ *with a set of locations* $L \subseteq \mathcal{L}$*; initial and time-can-progress predicates* $\mathrm{init}, \mathrm{tcp}\colon L \to \mathcal{P}$*; and a set of edges* $E \subseteq L \times \mathcal{P} \times \mathcal{A} \times L$.

Starting with atomic automata, *μCIF* automata are built using a set of compositions $C \in \mathcal{C}$, given by $C ::= \alpha \mid\ C \parallel_S C \mid\ \upsilon_U(C) \mid\ [\![\, x, \dot{x} \mapsto c, d :: C \,]\!]$, where α is an atomic automaton, $S \subseteq \mathcal{A}$, $U \subseteq \mathcal{A}$, $x \in \mathcal{V}$, $\dot{x} \in \dot{\mathcal{V}}$, and $c, d \in \Lambda$.

The semantics of *μCIF* automata is in terms of a hybrid transition system (HTS). Each state of this HTS is a pair $\langle p, \sigma \rangle$ comprising a composition of automata p and a *valuation* σ that associates a value to each variable. The set of all valuations is denoted by $\Sigma \triangleq \mathcal{V} \cup \dot{\mathcal{V}} \to \Lambda$. We keep track of the evolution of the values of variables using the concept of *variable trajectories*, denoted as $\rho : \mathcal{T} \to \Sigma$. A variable trajectory ρ holds the values of the variables at each time point $s \in \mathcal{T}$ in the delay. Since the values of variables change over time, the truth values of the guards dependent on the variable change as well. Thus, the set of enabled action transitions at each point in time can be determined using the concept of *guard trajectories.* A guard trajectory, denoted as $\theta : \mathcal{T} \to 2^{\mathcal{A}}$, keeps track of the set of enabled actions for each point in time.

The discrete and continuous behavior of HTSs is defined in terms of action transitions, given by $\_ \xrightarrow{\_} \_ \subseteq (\mathcal{C} \times \Sigma) \times \mathcal{A} \times (\mathcal{C} \times \Sigma)$, and timed transitions, given by $\_ \overset{\_}{\longmapsto} \_ \subseteq (\mathcal{C} \times \Sigma) \times ((\mathcal{T} \to \Sigma) \times (\mathcal{T} \to 2^{\mathcal{A}})) \times (\mathcal{C} \times \Sigma)$, respectively. The intuition of an action transition $\langle p, \sigma \rangle \xrightarrow{a} \langle p', \sigma' \rangle$ is that $\langle p, \sigma \rangle$ executes the discrete action $a \in \mathcal{A}$ and thereby transforms into $\langle p', \sigma' \rangle$, where p' and σ' denote the resulting automaton and variable valuation, respectively. The intuition behind a timed transition $\langle p, \sigma \rangle \overset{\rho,\theta}{\longmapsto} \langle p', \sigma' \rangle$ is that during the passage of time, the valuation that defines the values of the visible variables at each time-point $s \in [0, t] = \mathrm{dom}(\rho) = \mathrm{dom}(\theta)$ is given by the variable trajectory $\rho(s)$. The novelty in this paper is the set of enabled actions at each time-point $s \in [0, t]$, given by the guard trajectory $\theta(s)$. At the end-time point $t \in \mathcal{T}$, the resulting state is $\langle p', \sigma' \rangle$.

Structural Operational Semantics We use $\sigma(x)$, with $\sigma \in \Sigma$, to denote the value of variable x in valuation σ. By $\sigma \models u$ we denote that the predicate $u \in \mathcal{P}$ is satisfied in the valuation σ.

An atomic automaton $(L, \mathrm{init}, \mathrm{tcp}, E)$ can execute actions or time delays. Given an active location $\ell \in L$, i.e., a location for which $\mathrm{init}(\ell)$ holds, an action a

can be executed only if there is an edge (ℓ, g, a, ℓ') such that the guard g is satisfied. Rule 1 formalizes as follows:

$$\frac{(\ell, g, a, \ell') \in E,\ \sigma \models \mathrm{init}(\ell),\ \sigma \models g}{\langle (L, \mathrm{init}, \mathrm{tcp}, E), \sigma \rangle \xrightarrow{a} \langle (L, \mathrm{id}_{\ell'}, \mathrm{tcp}, E), \sigma \rangle}\ 1$$

where $\mathrm{id}_\ell \in L \to \mathcal{P}$ for $\ell \in \mathcal{L}$ such that $\mathrm{id}_\ell(\ell'') \triangleq\ \ell = \ell''$.

During a time delay $[0, t]$ for $t \in \mathcal{T}$, the values of variables can change. In a timed transition $\langle p, \sigma \rangle \overset{\rho, \theta}{\longmapsto} \langle p', \sigma' \rangle$, the variable trajectory ρ contains the values of variables at each point in the time interval $[0, t]$. The value of x can be defined as a function $\rho \downarrow x : [0, t] \to \Lambda$, in terms of ρ, such that $(\rho \downarrow x)(s) = \rho(s)(x)$.

In the CIF tooling, the variables x and $\dot{x}$ are, in principle, different variables. This makes it easier to implement an algorithm that checks whether a certain variable trajectory is a solution of an equation. The coupling between x and $\dot{x}$ is performed through the definition of a *dynamic type*. Most importantly, this dynamic type ensures that discrete variables remain constant over time, and that continuous variables have the expected derivatives. Formally, the dynamic type of a variable is a set $G \subseteq 2^{(\mathcal{T} \to \Lambda \times \mathcal{T} \to \Lambda)}$. A variable trajectory ρ is said to satisfy a dynamic type constraint G if $(\rho \downarrow x, \rho \downarrow \dot{x}) \in G$. For each variable x we assume the existence of a dynamic type G_x associated to it.

Now, timed transitions are enabled in an active location ℓ if there exists a *valid variable trajectory*. A variable trajectory ρ is valid if it has a positive duration, i.e. $\mathrm{dom}(\rho) = [0, t]$ and $0 < t$; tcp holds during $[0, t)$, with $[0, 0) \triangleq \emptyset$, and all variables satisfy the *dynamic type constraints*. The guard trajectory is constructed accordingly based on the variable trajectory, as given by rule 2:

$$\frac{\rho(0) \models \mathrm{init}(\ell), \forall_{s \in [0,t)}\, \rho(s) \models \mathrm{tcp}(\ell), \forall_{x \in \mathcal{V}}\, (\rho \downarrow x, \rho \downarrow \dot{x}) \in G_x}{\langle (L, \mathrm{init}, \mathrm{tcp}, E), \rho(0) \rangle \overset{\rho, \theta}{\longmapsto} \langle (L, \mathrm{id}_\ell, \mathrm{tcp}, E), \rho(t) \rangle}\ 2$$

where the duration of the delay is positive, $0 < t$, and the trajectories are defined exactly on the delay interval $\mathrm{dom}(\rho) = [0, t]$, $\mathrm{dom}(\theta) = [0, t]$. The guard trajectory is defined as $\forall_{s \in [0,t]}\, \theta(s) = \{a \mid (\ell, g, a, \ell') \in E \wedge \rho(s) \models g\}$.

Parallel composition As a result of composing two μCIF automata in parallel, the equally labeled action transitions of both components that are specified in the synchronization set $S \subseteq \mathcal{A}$ must be taken together. This is given by rule 3. The other action transitions are interleaved, as stated by rules 4 and 5.

$$\frac{\langle p, \sigma \rangle \xrightarrow{a} \langle p', \sigma' \rangle, \langle q, \sigma \rangle \xrightarrow{a} \langle q', \sigma' \rangle, a \in S}{\langle p \parallel_S q, \sigma \rangle \xrightarrow{a} \langle p' \parallel_S q', \sigma' \rangle}\ 3$$

$$\frac{\langle p, \sigma \rangle \xrightarrow{a} \langle p', \sigma' \rangle, a \notin S}{\langle p \parallel_S q, \sigma \rangle \xrightarrow{a} \langle p' \parallel_S q, \sigma' \rangle}\ 4 \qquad \frac{\langle q, \sigma \rangle \xrightarrow{a} \langle q', \sigma' \rangle, a \notin S}{\langle p \parallel_S q, \sigma \rangle \xrightarrow{a} \langle p \parallel_S q', \sigma' \rangle}\ 5$$

For timed transitions, the two components must agree in their variable trajectories (and hence in the duration of the time delay). The guard trajectory is constructed from the guard trajectories of the components of the parallel composition: at a given time point s, an action is enabled in the parallel composition

$p \parallel_S q$ if is enabled in p and q (regardless of whether the action is in S), or if it is enabled in p or q and it is not synchronizing. Rule 6 formalizes this:

$$\frac{\langle p, \sigma\rangle \overset{\rho,\theta_p}{\longmapsto} \langle p', \sigma'\rangle, \langle q, \sigma\rangle \overset{\rho,\theta_q}{\longmapsto} \langle q', \sigma'\rangle}{\langle p \parallel_S q, \sigma\rangle \overset{\rho,\theta_p \oplus_S \theta_q}{\longmapsto} \langle p' \parallel_S q', \sigma'\rangle}\ 6$$

where $(\theta_p \oplus_S \theta_q)(t) \triangleq (\theta_p(t) \cap \theta_q(t)) \cup (\theta_p(t) \setminus S) \cup (\theta_q(t) \setminus S)$.

Urgency operator The urgency operator $\upsilon_U(p)$ gives actions from the set $U \subseteq \mathcal{A}$ a higher priority than timed transitions. Timed transitions are allowed only if at the current state, and at each intermediate state while delaying, there is no urgent action enabled. This is given by rules 7 and 8 as follows:

$$\frac{\langle p, \sigma\rangle \xrightarrow{a} \langle p', \sigma'\rangle}{\langle \upsilon_U(p), \sigma\rangle \xrightarrow{a} \langle \upsilon_U(p'), \sigma'\rangle}\ 7 \quad \frac{\langle p, \sigma\rangle \overset{\rho,\theta}{\longmapsto} \langle p', \sigma'\rangle,\ \forall_{s\in[0,t)}\, U \cap \theta(s) = \emptyset}{\langle \upsilon_U(p), \sigma\rangle \overset{\rho,\theta}{\longmapsto} \langle \upsilon_U(p'), \sigma'\rangle}\ 8$$

Variable Scope We introduce local variables by means of the variable scope operator. The variable scope $[\![\, x, \dot{x} \mapsto c, d :: p \,]\!]$, for $x \in \mathcal{V}$, $\dot{x} \in \dot{\mathcal{V}}$, $c, d \in \Lambda$, and $p \in \mathcal{C}$, behaves as p but all the changes made to the local variables x and $\dot{x}$ are invisible. The initial values of x and $\dot{x}$ are c and d, respectively.

To define the semantics of this operator we use the function overwriting operator $\succ$, which, given two functions f and g, is defined as $f \succ g = f \cup g \downharpoonright_{\mathrm{dom}(f)}$, where $g \downharpoonright_X$ is the restriction of the domain of function g to $\mathrm{dom}(g) \setminus X$. In rules 9 and 10 the local value of the variables at the end of the transition is kept in the scope operator, whereas changes in the global valuation are overridden.

$$\frac{\langle p, \sigma_{xcd}\rangle \xrightarrow{a} \langle p', \sigma'\rangle}{\langle [\![\, x, \dot{x} \mapsto c, d :: p \,]\!], \sigma\rangle \xrightarrow{a} \langle [\![\, x, \dot{x} \mapsto \sigma'(x), \sigma'(\dot{x}) :: p' \,]\!], \sigma'_{xef}\rangle}\ 9$$

$$\frac{\langle p, \sigma_{xcd}\rangle \overset{\rho,\theta}{\longmapsto} \langle p', \sigma'\rangle}{\langle [\![\, x, \dot{x} \mapsto c, d :: p \,]\!], \rho_x(0)\rangle \overset{\rho_x,\theta}{\longmapsto} \langle [\![\, x, \dot{x} \mapsto \sigma'(x), \sigma'(\dot{x}) :: p' \,]\!], \rho_x(t)\rangle}\ 10$$

where $\forall_{\sigma\in\Sigma,\, x\in\mathcal{V},\, c,d\in\Lambda}\ \sigma_{xcd} \triangleq \{x \mapsto c, \dot{x} \mapsto d\} \succ \sigma$ and $\mathrm{dom}(\rho_x) = \mathrm{dom}(\rho)$ with $\forall_{s\in\mathrm{dom}(\rho)}\, \exists_{c,d\in\Lambda}\ \rho_x(s) \triangleq \rho(s) \downharpoonright_{\{x,\dot{x}\}} \cup \{x \mapsto c, \dot{x} \mapsto d\}$.

Stateless bisimilarity Two *µCIF* components are equivalent if they have the same behavior (in the bisimulation sense) given the same valuation of variables.

Definition 2. *A symmetric relation $R \subseteq \mathcal{C} \times \mathcal{C}$ is a stateless bisimulation relation if and only if for all $(p, q) \in R$ it holds that (1) $\forall_{\sigma,\sigma'\in\Sigma,\, a\in\mathcal{A},\, p'\in\mathcal{C}}\ \langle p, \sigma\rangle \xrightarrow{a} \langle p', \sigma'\rangle \Rightarrow \exists_{q'\in\mathcal{C}}\, \langle q, \sigma\rangle \xrightarrow{a} \langle q', \sigma'\rangle \wedge (p', q') \in R$, and (2) $\forall_{\sigma,\sigma'\in\Sigma,\, \rho\in\mathcal{T}\to\Sigma,\, \theta\in\mathcal{T}\to 2^{\mathcal{A}},\, p'\in\mathcal{C}}$ $\langle p, \sigma\rangle \overset{\rho,\theta}{\longmapsto} \langle p', \sigma'\rangle \Rightarrow \exists_{q'\in\mathcal{C}}\, \langle q, \sigma\rangle \overset{\rho,\theta}{\longmapsto} \langle q', \sigma'\rangle \wedge (p', q') \in R$. Two components p and q are stateless bisimilar, denoted by $p \underline{\leftrightarrow} q$, if there exists a stateless bisimulation relation R such that $(p, q) \in R$.*

Stateless bisimilarity is a congruence for all μCIF operators. This facilitates symbolic reasoning as we can replace equivalent automata in any context.

Theorem 1. *Stateless bisimilarity is a congruence over all μCIF operators.*

Proof. The SOS rules 1–10 satisfy the *process-tyft* format, which guarantees congruence for stateless bisimilarity [30]. $\square$

Urgency and variable abstraction The core of the problem in the interaction between urgency and variable abstraction is in the use of timed transition systems. In the previous subsections, we have given a semantics that does not abstract to a timed transition system, but rather to a 'guard trajectory labeled' transition system. Next, we give the main theorem of this paper, and prove that this change in semantics indeed solves the problem. The theorem states that urgency and variable abstraction distribute, meaning that the order in which they are applied is of no consequence anymore.

Theorem 2. $\forall_{p \in \mathcal{C},\, U \subseteq \mathcal{A}}\ [\![\ x, \dot{x} \mapsto c, d :: \upsilon_U(p)\ ]\!]\ \underline{\leftrightarrow}\ \upsilon_U([\![\ x, \dot{x} \mapsto c, d :: p\ ]\!])$.

Proof. Let $\mathcal{R} = \{([\![\ x, \dot{x} \mapsto c, d :: \upsilon_U(r)\ ]\!], \upsilon_U([\![\ x, \dot{x} \mapsto c, d :: r\ ]\!])) \mid r \in \mathcal{C}, U \subseteq \mathcal{A}, c, d \in \Lambda\}$. We will show that R is a stateless bisimulation, i.e., it satisfies the conditions of Definition 2.

Let $(p, q) \in R$ be such that $p = [\![\ x, \dot{x} \mapsto c, d :: \upsilon_U(r)\ ]\!]$ and $q = \upsilon_U([\![\ x, \dot{x} \mapsto c, d :: r\ ]\!])$. Assume $\langle [\![\ x, \dot{x} \mapsto c, d :: \upsilon_U(r)\ ]\!], \sigma\rangle \xrightarrow{a} \langle p', \sigma'_{xef}\rangle$. By rules 9 and 7, we know that $p' = [\![\ x, \dot{x} \mapsto \sigma'(x), \sigma'(\dot{x}) :: \upsilon_U(r')\ ]\!]$, $\langle \upsilon_U(r), \sigma_{xcd}\rangle \xrightarrow{a} \langle \upsilon_U(r'), \sigma'\rangle$, and $\langle r, \sigma_{xcd}\rangle \xrightarrow{a} \langle r', \sigma'\rangle$. Thus applying rules 9 and 7, we have that $\langle \upsilon_U([\![\ x, \dot{x} \mapsto c, d :: r\ ]\!]), \sigma\rangle \xrightarrow{a} \langle \upsilon_U([\![\ x, \dot{x} \mapsto \sigma'(x), \sigma'(\dot{x}) :: r'\ ]\!]), \sigma'_{xef}\rangle$.

Next suppose $\langle p, \rho_x(0)\rangle \overset{\rho_x, \theta}{\longmapsto} \langle p', \rho_x(t)\rangle$. By rule 10, we have that $\langle \upsilon_U(r), \sigma_{xcd}\rangle \overset{\rho, \theta}{\longmapsto} \langle \upsilon_U(r'), \sigma'\rangle$, and by rule 8 we have that $\langle r, \sigma_{xcd}\rangle \overset{\rho, \theta}{\longmapsto} \langle r', \sigma'\rangle$ and $\forall_{s \in [0,t)}\, U \cap \theta(s) = \emptyset$. Applying rule 10 to the last transition we get $\langle [\![\ x, \dot{x} \mapsto c, d :: r\ ]\!], \rho_x(0)\rangle \overset{\rho_x, \theta}{\longmapsto} \langle [\![\ x, \dot{x} \mapsto c, d :: r'\ ]\!], \rho_x(t)\rangle$, and since we know that $\forall_{s \in [0,t)}\, U \cap \theta(s) = \emptyset$, we can apply rule 8 to obtain $\langle \upsilon_U([\![\ x, \dot{x} \mapsto c, d :: r\ ]\!]), \rho_x(0)\rangle \overset{\rho_x, \theta}{\longmapsto} \langle \upsilon_U([\![\ x, \dot{x} \mapsto c, d :: r'\ ]\!]), \rho_x(t)\rangle$. $\square$

We note that an essential part of the proof of Theorem 2 is that the guard trajectories remain the same after variable abstraction, allowing us to deduce that urgent transitions do not interrupt the passage of time. Only when a system is finite set refutable[14], these guard trajectories can be recovered from the duration of the time transitions and the guards on the intermediate states.

4 Compositionality and Synchronization

We discuss the compositionality features of μCIF that are prominent in the literature, as well as passage of time when synchronizing urgent action transitions.

Compositionality of the parallel composition When synchronizing action transitions in timed and hybrid automata, one typically takes the conjunction of the guards involved in the synchronization as a guard of the synchronized transition. As noted in [20, 23] this is the most prominent way of synchronization. As an illustration, we depict the composition of two μCIF automata in Fig. 2, where $P_1 \parallel_{\{a\}} P_2 \underline{\leftrightarrow} P_3$. Recall that initial states are depicted by incoming arrows on which we state the initial values of the variables. State invariants like tcp predicates are placed inside a state.

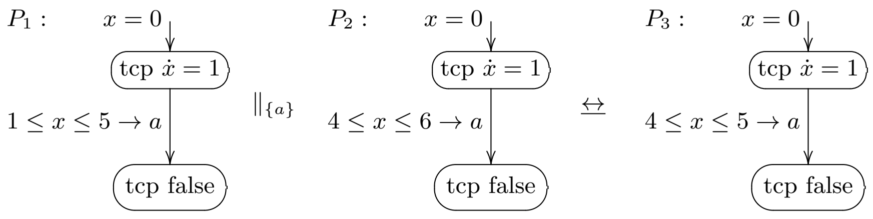


Fig. 2. Parallel composition with synchronization of μCIF automata

Taking the conjunction of the guards as a guard of the synchronized actions may lead to compositionality problems, depending on the way urgency is defined. As an example, consider the implementation of urgent transitions through deadlines, as in [20]. In Fig. 3 we have two timed automata with deadlines (TAD), T_1 and T_2. The initial value of the clock x is 0 and the guards $g : 1 \leq x \leq 5$ and $g : 4 \leq x \leq 6$ are associated with the actions a and c, respectively, denoting when the transitions may be taken. The deadlines $d : x \geq 3$ and $d : x \geq 6$ express when the transitions labeled by a and c must be taken, respectively. Thus, the action a is enabled in the interval $[1, 5]$ and must be taken at 3, whereas c is enabled in the interval $[4, 6]$ and must be taken at 6. The TAD T_1 and T_2 are considered bisimilar, since due to the urgency of the action a, the transition labeled by c will never be taken. However, when synchronizing on the action a, with a component that suppresses it, e.g., a component 'stop' that only idles, we see that the parallel composition behaves differently. Namely, the previously preempted action c is now observable in $T_2 \parallel_{\{a\}}$ stop, whereas $T_1 \parallel_{\{a\}}$ stop. Consequently, standard timed bisimulation [1] is not a congruence for TAD [20].

In μCIF, we solve the compositionality issue by defining the semantics of urgency and synchronization differently. We specify urgency using the tcp predicate, which is basically a state timed invariant, different from deadlines, which are on transition level. Additionally, we use SOS to define synchronization directly on the semantic level of transition systems, rather than defining it on the symbolic level of automata. In this way, we obtain compositionality for free by adhering to the process-tyft format from [30].

The closest mimic to the automata of Fig. 3 is given by the μCIF automata in Fig. 4, where the behavior of the automata T_1, T_2, and 'stop' is mimicked by

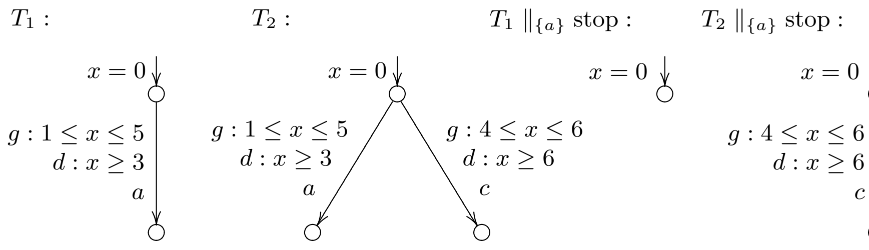


Fig. 3. Compositionality of timed automata with deadlines

P_4, P_5 and P_6, respectively. The crucial difference is that we use the tcp state predicate to stop the progress of time in order to induce urgency, which imposes a unique deadline for all outgoing transitions. This is observed in the initial state of P_5 as tcp $x \leq 3 \wedge x \leq 6$, which is equivalent to tcp $x \leq 3$. In our setting, P_4 is bisimilar to P_5, but the change in the semantics of urgency and the parallel composition ensures that $P_4 \parallel_{\{a\}} P_6$ is also bisimilar to $P_5 \parallel_{\{a\}} P_6$.

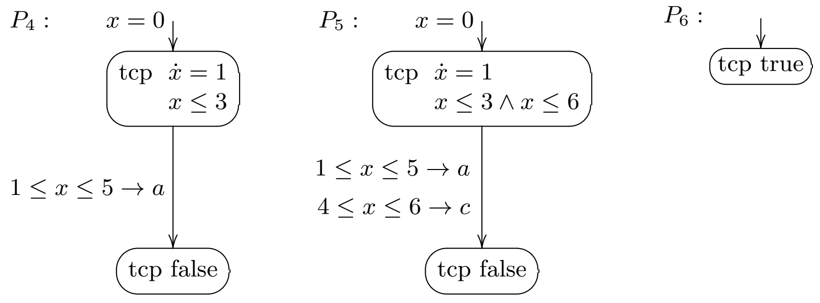


Fig. 4. *µCIF* automata P_4, P_5, and P_6, mimicking the behavior of TAD automata T_1, T_2, and stop, of Fig. 3, respectively.

Impatient and patient synchronization Above we showed how to impose hard deadlines by means of the tcp predicate. Alternatively, one could obtain the same behavior by means of the global urgency operator υ_U for $U \subseteq \mathcal{A}$. We put the automaton on which we want to impose urgency in parallel with an automaton that request urgency on synchronizing action transitions as depicted in Fig. 5. Here, we show how to alternatively specify the urgency of action a at time 3, obtaining *µCIF* automaton P_7 that is bisimilar to the automaton P_4 of Fig. 4.

An advantage of the specification of deadlines using the urgency operator, is that it provides for more flexibility. In particular, the scope of the urgency operator can be used to specify both impatient and patient synchronization of actions. In impatient synchronization, we have hard deadlines. Therefore, the

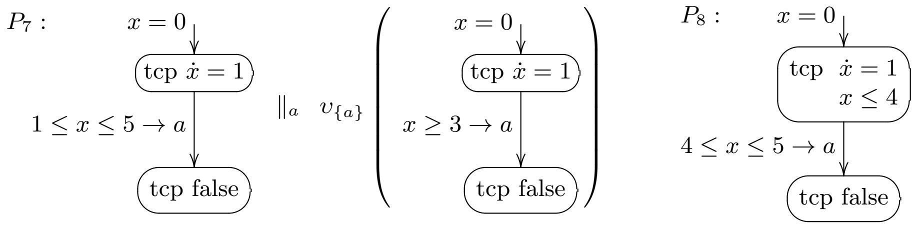


Fig. 5. The *μCIF* automaton P_7 shows an alternative definition of hard deadlines $P_7 \underline{\leftrightarrow} P_4$; the *$\mu$CIF* automaton P_8 is the result of the patient synchronization of P_1 and P_2, i.e., $P_8 \underline{\leftrightarrow} \upsilon_{\{a\}}(P_1 \parallel_{\{a\}} P_2)$.

urgency constraints of all synchronizing parties must be endorsed as soon as they are enabled. In patient synchronization we have soft deadlines. Some urgency constraints can be contravened so that the synchronization can occur as long as the guards still hold. To specify patient synchronization, we place the parallel composition *inside* the scope of the urgency operator. For example, $\upsilon_{\{a\}}(P_1 \parallel_{\{a\}} P_2)$, where P_1 and P_2 are given in Fig. 2 specifies the patient synchronization of P_1 and P_2. The resulting system has an outgoing guarded action transition $4 \leq x \leq 5 \to a$, which must be taken at time 4, restricted by tcp $x \leq 4$.

The urgency operator we defined, provides a reasonable amount of flexibility at specifying various types of synchronization. Ongoing research shows that the global urgency operator can be eliminated through symbolic reasoning at the syntactic level, by transferring it to local urgency of the tcp predicate. We can achieve this by means of a linearization procedure that transforms *μCIF* automata by pushing the guards of the urgent transitions in the tcp predicate. As a consequence, guard trajectories never need to be actually calculated when proving properties of a system. Their only purpose is to retain compositionality when combining urgency with variable hiding. After a composition has been linearized, the need for guard trajectories disappears. The linearization approach can be illustrated by observing the *μCIF* automata P_4 of Fig. 4 and P_7 of Fig. 5. The guard of the urgent transitions a of P_7, given by $x \geq 3$, is pushed to the tcp predicate of the initial state of P_4, given by tcp $x \leq 3$. Thus, the linearization procedure provides an efficient implementation of urgency in the CIF toolset [13].

5 Concluding remarks

In this paper, we investigated the interplay between urgent actions and variable abstraction in timed and hybrid systems. We showed that this interaction is not distributive due to the use of timed transition systems as the basic semantic model, rendering component-wise verification inapplicable. We proposed to add guard trajectories in the labels of the timed transitions as a remedy. We illustrated the proposal by revising the semantics of the *CIF* language, as it is used in the MULTIFORM project. We proved that the identified problem indeed disappears, while retaining commonly desired properties of urgency and variable abstraction, such as compositionality and the possibility of specifying different kinds of synchronization. Our approach employs SOS techniques, guaranteeing compositionality with respect to stateless bisimilarity. Furthermore, it makes that the concepts introduced in this paper easily transferrable to other timed and hybrid formalisms. This has, e.g., already been done for the χ language [10]. Whether our results are useful in the verification of hybrid systems with urgency, remains as a topic for future research. Ongoing research based on linearization procedures that eliminate the urgency operator is promising and, moreover, the obtained results provide for local variable abstraction and verification based on this abstraction, an option not available in previous work.

Acknowledgements: We thank Albert Hofkamp for his original idea to introduce guards in the transition system, Ramon Schiffelers for his early involvement in this paper, and Mohammad Mousavi for his advise on the SOS.

References

1. Alur, R., Dill, D.: A theory of timed automata. Theoretical Computer Science **126** (1994) 183–235
2. Larsen, K., Pettersson, P., Yi, W.: UPPAAL in a nutshell. International Journal on Software Tools for Technology Transfer **1**(1-2) (1997) 134–152
3. Bozga, M., Daws, C., Maler, O., Olivero, A., Tripakis, S., Yovine, S.: Kronos: A model-checking tool for real-time systems. In: CAV 1998. Volume 1427 of Lecture Notes in Computer Science., Springer (1998) 546–550
4. Bozga, M., Mounier, L., Graf, S.: If-2.0: A validation environment for component-based real-time systems. In: CAV 2002. Volume 2404 of Lecture Notes in Computer Science., Springer (2002) 343–348
5. Henzinger, T.: The theory of hybrid automata. In: Verification of Digital and Hybrid Systems. Volume 170 of NATO ASI Series F: Computer and Systems Science. Springer-Verlag (2000) 265–292
6. Lynch, N., Segala, R., Vaandrager, F.: Hybrid I/O automata revisited. In: HSCC 2001. Volume 2034 of Lecture Notes in Computer Science., Springer (2001) 403–417
7. Henzinger, T., Ho, P.H., Wong-toi, H.: HyTech: A model checker for hybrid systems. International Journal on Software Tools for Technology Transfer **1**(1) (1997) 110–122
8. Lee, E., Zheng, H.: Operational semantics of hybrid systems. In: HSCC 2005. Volume 3414 of Lecture Notes in Computer Science., Springer (2005) 25–53
9. Reynolds, J.C.: Theories of programming languages. Cambridge University Press, New York, NY, USA (1999)
10. Beek, D.A.v., Man, K.L., Reniers, M.A., Rooda, J.E., Schiffelers, R.R.H.: Syntax and consistent equation semantics of hybrid Chi. Journal of Logic and Algebraic Programming **68**(1-2) (2006) 129–210
11. Plotkin, G.: A structural approach to operational semantics. Technical Report DAIMI FN-19, Computer Science Department, Aarhus University (1981)
12. Beek, D.A.v., Collins, P., Nadales, D.E., Rooda, J., Schiffelers, R.R.H.: New concepts in the abstract format of the compositional interchange format. In: ADHS 2009, IFAC (2009) 250–255

13. Systems Engineering Institute: The Compositional Interchange Format for Hybrid Systems. http://se.wtb.tue.nl/sewiki/cif/start (2010)
14. Cuijpers, P., Reniers, M.: Lost in translation: Hybrid-time flows vs real-time transitions. In: HSCC 2008. Volume 4981 of Lecture Notes in Computer Science., Springer (2008) 116–129
15. Alur, R., Henzinger, T.: Real-time system = discrete system + clock variables. In: Theories and Experiences for Real-time System Development, AMAST Series in Computing 2, World Scientific (1993) 1–29
16. Kaynar, D., Lynch, N., Segala, R., Vaandrager, F.: A framework for modelling timed systems with restricted hybrid automata. In: RTSS 2003, IEEE Computer Society Press (2003) 166–178
17. Bornot, S., Sifakis, J., Tripakis, S.: Modeling urgency in timed systems. In: COMPOS 1998. Volume 1536 of Lecture Notes in Computer Science., Springer (1998) 103–129
18. Sifakis, J.: The compositional specification of timed systems. In: CAV 1999. Volume 1633 of Lecture Notes in Computer Science., Springer (1999) 2–7
19. Bornot, S., Sifakis, J.: An algebraic framework for urgency. Information and Computation **163** (2000) 172–202
20. D'Argenio, P., Gebremichael, B.: The coarsest congruence for timed automata with deadlines contained in bisimulation. In: CONCUR 2005. Volume 3653 of Lecture Notes in Computer Science., Springer (2005) 125–140
21. Bowman, H.: Modeling timeouts without timelocks. In: ARTS 1999. Volume 1601 of Lecture Notes in Computer Science., Springer (1999) 20–35
22. Bohnenkamp, H., D'Argenio, P., Hermanns, H., Katoen, J.P.: MODEST: A compositional modeling formalism for hard and softly timed systems. IEEE Transactions on Software Engineering **32**(10) (2006) 812–830
23. Gebremichael, B., Vaandrager, F.: Specifying urgency in timed I/O automata. In: SEFM 2005, IEEE Computer Society (2005) 64–74
24. Mufti, W., Tcherukine, D.: Integration of model-checking tools: from discrete to hybrid models. In: INMIC 2007, IEEE International (2007) 1–4
25. Bornot, S., Sifakis, J.: On the composition of hybrid systems. In: HSCC 1998. Volume 1386 of Lecture Notes in Computer Science., Springer (1998) 49–63
26. van Beek, D., Hofkamp, A., Reniers, M., Rooda, J., Schiffelers, R.: Syntax and formal semantics of Chi 2.0. SE Report 2008-01, Eindhoven University of Technology (2008)
27. Cuijpers, P., Reniers, M., Heemels, W.: Hybrid transition systems. Technical Report CS-Report 02-12, TU/e, Eindhoven, Netherlands (2002)
28. Beek, D.A.v., Reniers, M.A., Schiffelers, R.R.H., Rooda, J.E.: Foundations of an interchange format for hybrid systems. In: HSCC 2007. Volume 4416 of LNCS., Springer (2007) 587–600
29. Scott, D.: Outline of a mathematical theory of computation. In: CISS 1970, Princeton (1970) 169–176
30. Mousavi, M.R., Reniers, M.A., Groote, J.F.: Notions of bisimulation and congruence formats for SOS with data. Information and Computation **200**(1) (2005) 107–147