Systems Engineering Group
Department of Mechanical Engineering
Eindhoven University of Technology
PO Box 513
5600 MB Eindhoven
The Netherlands
http://se.wtb.tue.nl/

SE Report: Nr. 2006-05

Foundations of a compositional interchange format for hybrid systems

D.A. van Beek M.A. Reniers
R.R.H. Schiffelers J.E. Rooda

ISSN: 1872-1567

SE Report: Nr. 2006-05
Eindhoven, November 2006
SE Reports are available via http://se.wtb.tue.nl/sereports

Abstract

A compositional interchange format for hybrid systems is defined in terms of an interchange automaton, allowing arbitrary differential algebraic equations, including fully implicit or switched DAEs, discrete, continuous and algebraic variables, that can be internal or external, urgency conditions, and operators for parallel composition, action hiding, variable hiding and urgent actions. Its compositional semantics is formally defined in terms of a hybrid transition system. This allows development of transformations to and from other formalisms that can be proven to preserve essential properties, and it allows a clear separation between the mathematical meaning of a model and implementation aspects such as algorithms used for solving differential algebraic equations.[1]

[1]Work partially done in the framework of the HYCON Network of Excellence, contract number FP6-IST-511368

Contents

1 Introduction

Our intention is to establish inter-operability of a wide range of tools by means of model transformations to and from a compositional interchange format that is defined in terms of an *interchange automaton*. The domain of the interchange automaton format consists of languages and tools from computer science and from dynamics and control for modeling, simulation, analysis, controller synthesis, and verification in the area of hybrid and timed systems. The purpose of an interchange format is to avoid the implementation of many bi-lateral translators between specific formalisms. Instead, the translation from a formalism A to a formalism B is divided in two steps: first, the model in formalism A is translated into a representation (model) in the interchange automaton format, then, this representation is translated into a model in formalism B [32].

Our main requirements for the interchange format are summarized below. A more detailed discussion of these requirements follows in Sections 2 and 3.

1. It should have a formal and compositional semantics, based on (hybrid) transition systems, and allow property preserving model transformations.
2. Its concepts should be based on mathematics, and independent of implementation aspects such as equation sorting, and numerical equation solving algorithms.
3. It should support arbitrary differential algebraic equations (DAEs), including fully implicit equations, higher index systems, algebraic loops, steady state initialization, switched systems such as piecewise affine systems, and DAEs with discontinuous right hand sides.
4. It should support a wide range of concepts originating from hybrid automata, including different kinds of urgency, such as 'urgency predicates', 'deadline predicates', 'triggering guard semantics', and 'urgent actions'.
5. It should support parallel composition with synchronization by means of shared variables and shared actions.
6. It should support hierarchy and modularity to allow the definition of parallel modules and modules that can contain other modules (hierarchy), and to allow the definition of variables and actions as being local to a module, or shared between modules.

Other work on interchange formats for hybrid systems has been carried out in different projects: in the MoBIES project, the Hybrid System Interchange Format (HSIF) [27] is defined; in [30] an 'abstract semantics' of an interchange format based on the Metropolis meta model is defined; this work is a continuation of the COLUMBUS project [10]; and in the HYCON NoE [21], an interchange format for switched linear systems [9] in the form of piecewise affine systems (PWAs) is defined.

In HSIF, a network of hybrid automata is used for model representation. The network behaves as a parallel composition of its automata, without hierarchy or modules. Variables can be shared or local, and the communication mechanism is based on broadcasting of boolean 'signals', where signals are partitioned in input and output signals. Each signal is required to be either a global input to the network or to be modified by exactly one automaton. The semantics is defined only for 'acyclic dependency graphs' with respect to the use of signals. The time dependent behavior is specified by means of ordinary differential equations (ODEs), together with algebraic relations of the form $x = f(x_1, \ldots, x_n)$, and invariants. The equation $\dot{x} = 0$ is assumed for each shared variable. Circular dependencies of the algebraic equations, i.e. algebraic loops, are not allowed. To provide for 'synchronous execution' of discrete steps, each automaton has a default transition. There are no algebraic variables, and there is no form of urgency predicate or urgent action. Execution of the network is defined as an alternating sequence of continuous steps followed by discrete steps, where each discrete step of the network is the result of a sequence of discrete steps of each of the automata taken in an order that respects automata dependencies [27]. The interchange automaton format defined in this article aims to be more general than HSIF, and does

not incorporate tool limitations, such as restrictions on circular dependencies or algebraic loops, in its compositional formal semantics.

The 'abstract semantics' presented in [30], takes implementation considerations into account, such as equations sorting, iterations that may be required for state-event detection, and iterations for reaching a fixed-point in case of algebraic loops. The semantics is defined in terms of functions and algorithms such as init, markchange, and solve. This is different from the compositional formal semantics as defined in Section 5, which aims at defining the *mathematical* meaning of interchange automata, independently of implementation aspects such as equation sorting or state-event detection. For example, the semantics defines the mathematical meaning of a switched system of equations, such as a PWA system, but an implementation may choose to implement such switching behavior with or without state-event detection.

A transformation from the PWA-based interchange format [9] to the interchange automaton format will be developed. Based on this transformation, several tools, based on among others PWA, HYSDEL, MLD (see [17] for an overview relating these languages) can then be connected to the interchange automaton format.

In principle, an interchange automaton can be specified in three different formats: First, an abstract format. The purpose of this format is to facilitate definition of the formal semantics. This is the format defined in this article. Second, a concrete format. The purpose of this format is to provide user friendly syntax, that can be used for modeling directly in the interchange automaton format. Third, a transfer format. The purpose of this format is to facilitate the file generation and parsing process. This format can, for example, be specified in an extensible markup language (XML) format, that is supported by means of libraries in many different languages.

The remainder of this article is organized as follows: Section 2 discusses the importance of a compositional formal semantics, Section 3 discusses the concepts present in the interchange automaton format, Sections 4 and 5 define the syntax and semantics of interchange automata, respectively, Section 6 shows how the operators of interchange automata can be eliminated resulting in atomic interchange automata, and Section 7 presents concluding remarks.

2 Importance of a compositional formal semantics

A formal semantics defines the mathematical meaning of a valid model given in the interchange automaton format, i.e., it defines all possible behaviors of the model for certain initial conditions. A formal semantics allows a clear distinction between the mathematical meaning of a model and implementation aspects. It defines a standard against which implementations, such as simulation or verification implementations can be evaluated. In order to use the interchange automaton format for verification purposes, translations from models to the interchange automaton format, and vice versa, should preserve essential properties. I.e., verification results obtained for a derived model should also be valid for the original model specified in another formalism.

The different formalisms may have different features and semantics, complicating translations between them. In order to keep the translations between the interchange automaton format and the different formalisms manageable, in terms of complexity, it is essential that the interchange automaton format allows transformations of (parts of) specifications in the interchange automaton format itself. For instance formalism A could use a triggering guard semantics (see Section 3), no invariants, and hierarchy. Formalism B could use invariants, no concepts for urgency, and no hierarchy. The transformations of the triggering guards to invariants, and flattening of the model should all take place in the interchange automaton format.

To allow proof of equivalence (see Section 5.3) between the transformed model in the interchange automaton format and its original, it is essential that the semantics of the interchange automaton format is *compositional*, i.e., that the notion of equivalence is a congruence for all operators of the interchange automaton format. Parts of a model can then be replaced by equivalent parts without changing the meaning of the model.

For simulation of hybrid systems based on nonlinear DAEs, usually numerical solvers are used [8]. Numerical solvers can be parameterized with, among others, different settings for the relative and/or absolute accuracies. This means that simulation of the same model using different simulators, or using different solvers or different parameter settings for the same simulator, will usually lead to different results. In such cases, there will obviously be a difference between the formal semantics, which mathematically defines the allowed behaviors of the model, and the result of a simulation run, which gives a numeric approximation of one of the allowed behaviors. The formal semantics then defines the 'reference behavior' which can be used to evaluate the quality of the numeric approximation provided by a simulator.

The fact that a formal semantics defines the mathematical meaning of a model is also useful when dealing with advanced simulators for complex nonlinear DAEs. Several techniques exist for efficient simulation of such DAEs, such as ordering of the equations in block lower triangular structure, tearing, and reduction of the number of equations and unknowns by symbolic manipulation and solving [26]. Clearly, these techniques for improving simulation efficiency should not affect the simulation results. A formal semantics unambiguously defines the mathematical meaning of a model, and is thus completely independent of techniques used by advanced simulation algorithms.

3 Concepts in the interchange automaton format

3.1 Differential algebraic equations

Modeling of physical systems, such as mechanical or chemical systems frequently leads to DAEs. Algebraic constraints can also be the result of stateless components such as proportional controllers. DAEs can be modeled and simulated using languages such as Modelica [35] and EcosimPro [11]. DAEs can be specified in the invariants of an interchange automaton, since such invariants are predicates over all variables, including the dotted variables. Flow clauses are supported for reasons of compatibility with existing hybrid automata. The reason for not enforcing a separation between invariants (over non-dotted variables) and flow clauses (over dotted variables), as in existing hybrid automata, is that such a separation is absent in the mathematical theory of dynamical systems, including control theory. In many cases, fully implicit DAEs, such as $\mathbf{f}(\dot{\mathbf{x}}, \mathbf{x}, \mathbf{y}, t) = \mathbf{0}$, cannot even be rewritten to a form where the algebraic constraints and the differential constraints are separated, such as the semi-explicit form $\dot{\mathbf{x}} = \mathbf{g}(\mathbf{x}, \mathbf{y}, t)$, $\mathbf{h}(\mathbf{x}, \mathbf{y}, t) = \mathbf{0}$, where $\mathbf{x}$ and $\mathbf{y}$ are the continuous and algebraic variables, respectively. The generalized invariant allows us to consider the four expressions $x = 1 \wedge x = 2$, $\dot{x} = 1 \wedge \dot{x} = 2$, $\dot{x} = y \wedge \dot{x} = 2y \wedge y = 1$ and 'false' to be equivalent (bisimilar): no behavior is possible.

The initialization clause of the interchange automaton is also defined as a predicate over all variables, including the dotted variables. This allows more general initializations than usually allowed in hybrid automata. In particular, steady state initialization, as available in Modelica and EcosimPro, is supported. E.g. by defining $\dot{\mathbf{x}} = \mathbf{0}$ as initialization predicate for a location with invariant $\mathbf{f}(\dot{\mathbf{x}}, \mathbf{x}, \mathbf{y}, t) = \mathbf{0}$, the initial state is defined as the 'steady state', that is the solution of the set of DAEs such that all derivatives are zero: $\mathbf{f}(\mathbf{0}, \mathbf{x}, \mathbf{y}, t) = \mathbf{0}$.

3.2 Discrete, continuous and algebraic variables

The interchange automaton defines three classes of variables: the discrete and continuous variables, and in addition the algebraic variables. The differences are as follows: Continuous variables are the only variables for which dotted variables (derivatives) can be used in models. The values of discrete variables remain constant when model time progresses, the values of continuous variables may change according to a continuous function of time when model time progresses, and the values of algebraic variables may change according to a discontinuous function of time. Finally, there is a difference between the different classes of variables with respect to how the resulting values of the variables in a transition relate to the starting values of the variables in the

next transition. The resulting value of a discrete or continuous variable in a transition always equals its starting value in the next transition. For algebraic variables there is no such relation, because algebraic variables are not part of the state.

The state of an interchange automaton consists of, among others, the interchange automaton itself, and a valuation of the discrete and continuous variables (see Section 5 for a more precise definition of the state). The values of the dotted variables and the algebraic variables are not contained in the state. The reason for this is that the state of an interchange automaton represents all information needed to determine future behavior, i.e., the state of a system makes the system's history irrelevant. The dotted and algebraic variables are not needed in the state, because their values are determined completely by the interchange automaton: in particular by the initial conditions, the flow conditions, the invariants and the jump predicates as defined in Section 4.

In most languages that allow (implicit) DAEs, such as Modelica [35], EcosimPro [11], and Simulink [34], the distinction between continuous and algebraic variables is implicitly made by considering all continuous variables that do not occur differentiated as algebraic. EcosimPro also implicitly makes the distinction between discrete and continuous variables: variables of type real that occur differentiated are continuous (unless the variable is prefixed by the DISCR qualifier, in which case it is discrete). Variables of other types (such as integer, boolean and string) are discrete.

3.3 Automata related concepts

Many different hybrid automaton definitions exist. Some definitions require solutions for the continuous variables to be differentiable functions, e.g. in [19, 1]. Other definitions allow the more general case of piecewise differentiable or piecewise continuous functions, e.g. in [33]. Such restrictions can be realized in the interchange automaton format by means of the parameters F and G as defined in Section 5. In [24], for each variable a 'dynamic type' can be defined. However, since we did not find such expressivity in tools, the interchange automaton format allows the definition of the dynamic type for the algebraic and continuous variable classes, not for each individual variable.

With respect to the meaning of jump predicates, that define the behavior of the variables in action transitions, differences also occur: in [19] the variables can in principle perform arbitrary jumps unless restricted by the jump predicate, in [20], variables in principle remain unchanged unless changes are enforced by the jump predicate by means of primed variables. The first behavior is obtained by an interchange automaton that defines the set of jumping variables W (see Sections 4 and 5.1) at each edge to be equal to the set of all variables. The second kind of behavior is obtained by defining the set W as the union of all primed variables of the jump predicate. The specification of a set of jumping variables and a jump predicate for each edge of an interchange automaton is based on [1].

The interchange automaton format is expressive enough to deal with verification tools such as PHAVER [13] and HYTECH [20]. The behavior of the algebraic variables from the interchange automaton is related to the external variables from the semantical hybrid I/O automaton defined in [24]. In this I/O automaton, the external variables are also not part of the state, and they can have a dynamic type that allows discontinuous trajectories. The state is defined by the values of the internal variables, and discrete transitions (action transitions) are defined only on internal variables. The interchange automaton format can express this as a special case, since the different classes of variables, action transitions, and hiding/abstraction are orthogonal concepts in the interchange automaton format.

3.4 Urgency

The concept of urgency allows the passing of time up to a certain point. There are essentially two different kinds of urgency:

1. Urgency that is defined for an atomic automaton by means of one or more predicates. Such

predicates can be associated to a location, or to outgoing edges of the location.

2. Urgency that is defined as an operation on a composition of one or more automata. Such an operation defines a set of actions as urgent for the composition. The operation allows the passing of time up to the point when one or more of the urgent actions can be executed.

The first kind of urgency is defined in many different forms. The `tcp` (*time can progress*) predicate [29], is a predicate over the variables of the automaton and time. The predicate is associated to a location. It allows passing of time in a location for as long as the predicate is true. Related to the `tcp` predicate is the *stopping condition* [14], which is a predicate on the variables of the automaton, also associated to a location, and which allows passing of time in a location for as long as the stopping condition is false, or in other words, until the time-point when the stopping condition is true. *Deadline predicates* [7] and *urgency predicates* [14] are associated to the edges of an automaton. Deadline predicates allow passing of time in a location until the time-point that one or more deadline predicates of the outgoing edges of the location become true. Whenever a deadline predicate of an edge becomes true, the guard associated to that edge must also be true: the deadline predicate must imply the guard. Urgency predicates are similar to deadline predicates; the only difference is that they do not have the restriction that the urgency predicate should imply the guard. Urgency predicates allow passing of time in a location until the point of time that for one or more of the outgoing edges, the guard and the urgency predicate are both true.

Restricting a `tcp` predicate as a predicate over the variables of an automaton makes it equal to the negation of a stopping condition. Deadline predicates and urgency predicates are less expressive. They can both be expressed in terms of stopping conditions, see [14], or as `tcp` predicates. E.g. the stopping condition of a location corresponds to the disjunction of all deadline predicates of the outgoing edges of the location. Note that a flow condition which is false in a hybrid automaton is equivalent to a stopping condition that is true, or a `tcp` predicate that is false. The interchange automaton format adopts stopping conditions, which we refer to as *urgency conditions*.

In simulation languages, such as Modelica, EcosimPro, and HyVisual, usually a triggering guard semantics is used, meaning that the passing of time in a location is allowed until the time-point that any of the guards of the outgoing edges becomes true. This is equivalent to a stopping condition associated to the location that is the disjunction of the guards of all outgoing edges.

The second kind of urgency, as for example defined in [6] and [3], is often available with restrictions only. E.g. in HYTECH, edges can be defined as urgent. The composition of urgent actions is required to be well-formed: 'whenever two components synchronize on a label, if one transition is urgent then the other must either be urgent, or have a jump condition expressible as a guarded command with its guard being either the predicate true or the predicate false' [20]. A second restriction is that 'if there exists an urgent transition from a location v to a location v', then for all valuations satisfying the invariant of v, an urgent transition to v' should exist'. The timed automaton language UPPAAL [23] appears to define urgent communication by allowing the definition of channels either as urgent or non-urgent, but the semantics can be defined in terms of stopping conditions: time can pass in a location for as long as 1) none of its edges synchronize via an urgent channel, and 2) for all edges that synchronize via an urgent channel the guards remain false [36].

The interchange automaton format defines the second kind of urgency by means of the urgent action operator. Note that defining this kind of urgency by means of directly referring to the guards of actions may lead to congruence problems, as described in [5]. There, replacing a part of a specification by another part with the same behavior may lead to different behavior of the complete system.

3.5 Synchronous systems related concepts

An overview of the synchronous approach, as adopted by the three synchronous languages Esterel, Lustre and Signal, is given in [4]. Essential to this approach is the division of time into discrete instants and the distinction of inputs and outputs of a system. Execution of a synchronous model is, in principle, a deterministic transformation, at each time-instant, of the values for

each of the inputs and internal state, to the values of the outputs and the internal state. A characteristic difference with hybrid automaton related formalisms is the semantics of parallel composition. Hybrid automaton related formalisms usually have an interleaving, non-deterministic semantics for the execution of actions. Synchronous languages in principle define the behavior of parallel composition of input/output systems as the (deterministic) conjunction of the behaviors. In practice, however, such synchronous behavior is difficult to achieve, and many different semantics exist, ranging from true synchronous behavior to full interleaving. Interaction in synchronous systems is usually based on shared variables and/or 'signal broadcasting'. In [22], a formal semantics of timed and hybrid statecharts is presented. Signal broadcasting is formally defined by means of 'volatile variables'. In Harel's Statecharts [16], the semantics is defined in terms of 'steps' and 'supersteps'. In other languages these may be referred to as 'microsteps' and 'steps'. Many, possibly conflicting, definitions of microsteps based semantics exist [4]. The comparative study [37] discusses several issues of the Statechart semantics and treats more than twenty Statechart variants. Other languages that have been influenced by synchronous systems are Masaccio [18] and, recently also Charon [2]. Because of the many different ways of dealing with 'synchronous behavior', the synchronous approach needs to be studied in more detail before incorporating it in the interchange automaton.

The synchronous approach is to some extent already present in the interchange automaton, since the parallel composition of interchange automata takes as the behavior for time transitions the conjunction of the behaviors of the individual automata. Furthermore, the interchange automaton shares the *substitution principle* [15] with Lustre: an equation $x = e$, where x is a variable and e an expression, defined in the invariant of a single location automaton without edges, allows the substitution of x by its defining expression e everywhere in automata that are placed in parallel (assuming that the automata share variable x and the variables of expression e). This is referred to as the *consistent equation semantics* in [3]. Furthermore, each connection between an input and output of synchronous models, including Simulink [34] blocks, can be modeled by means of an algebraic variable in an interchange automaton. The function f of the block, say $\mathbf{y} = f(\mathbf{x}, \mathbf{u})$, where $\mathbf{x}$ is the state, $\mathbf{y}$ is the output, and $\mathbf{u}$ is the input, is modeled by means of an equation in the invariant of the interchange automaton.

4 Abstract syntax of interchange automata

Notation 4.1. The following notations are defined:

- A set $\mathcal{V}$ of variables and a set of action labels $\mathcal{L}$, which does not include the predefined non-synchronizing action τ, are assumed. The set $\mathcal{L}_\tau$ denotes the set $\mathcal{L} \cup \{\tau\}$.
- For a set of variables $S \subseteq \mathcal{V}$, $\dot{S} = \{\dot{x} \mid x \in S\}$ denotes the set of dotted variables.
- For a set of variables $S \subseteq \mathcal{V}$, Pred(S) denotes the set of all predicates over variables from S.
- $f : A \mapsto B$ and $g : A \to B$ define a partial function f and a total function g, both with domain A and range B.

Definition 4.2 (Atomic Interchange Automaton)**.** An *atomic interchange automaton* is a tuple $(X, X_\mathrm{i}, \mathrm{dtype}, V, v_0, \mathrm{init}, \mathrm{flow}, \mathrm{inv}, \mathrm{urgent}, L, E)$ where

- $X \subseteq \mathcal{V}$ is a finite set of variables, $X_\mathrm{i} \subseteq X$ is the set of *internal* variables, and $X_\mathrm{e} = X \setminus X_\mathrm{i}$ is the set of *external* variables.
- $\mathrm{dtype} : X \to \{\mathrm{disc}, \mathrm{cont}, \mathrm{alg}\}$ is a function that associates to each variable a dynamic type: *discrete*, *continuous* or *algebraic*. The sets X_disc, X_cont, X_alg are defined as $X_t = \{x \in X \mid \mathrm{dtype}(x) = t\}$ for $t \in \{\mathrm{disc}, \mathrm{cont}, \mathrm{alg}\}$, and $X_\mathrm{state} = X_\mathrm{disc} \cup X_\mathrm{cont}$ is the set of *state* variables.

- V is a finite non-empty set of *vertices*, called *locations*, and $v_0 \in V$ is the initial location.
- init $\in \text{Pred}(\tilde{X})$ is the initial condition. For $Y \subseteq X$, $\tilde{Y} = Y \cup \{\dot{y} \mid y \in Y \cap X_{\text{cont}}\}$ is the extension of Y with the dotted versions of the continuous variables in Y.
- flow, inv, urgent : $V \to \text{Pred}(\tilde{X})$, are functions that each associate to each location $v \in V$ a predicate describing the *flow condition*, the *invariant*, and the *urgency condition*, respectively.
- $L \subseteq \mathcal{L}$ is a finite set of action labels.
- $E = V \times \text{Pred}(\tilde{X}) \times (L \cup \{\tau\}) \times (\mathcal{P}(\tilde{X}) \times \text{Pred}(\tilde{X} \cup \tilde{X}^-)) \times V$ is a finite set of *edges*, such that for each element $(v, g, l, (W, r), v') \in E$, v and v' are the *source* and *target* locations, respectively, g is the *guard*, l is the *action label*, $W \subseteq \tilde{X}$ is a set of jumping variables (the value of which may change as a result of an action transition), and r is the *jump predicate*, also called *reset map*. For any $Y \subseteq \tilde{X}$, $Y^- = \{y^- \mid y \in Y\}$ denotes the set of minus superscripted variables that represent the values of variables before an action transition.

Note that the *dynamic* type of a variable gives information about its time dependent behavior. E.g. the value of a discrete variable remains constant when time passes, whereas a the value of a continuous variable changes as a continuous function of time. The *static* type, such as real, integer or boolean, mainly gives information about the domain in which the variable takes values.

The interchange automaton format consists of automata, and operators for parallel composition, for hiding of action labels and variables, and for the definition of urgent actions. The automata and operators can be freely combined:

Definition 4.3 (Interchange automaton)**.** The set of interchange automata A is defined by the following grammar for the interchange automata $\alpha \in A$:

α	::=	α_{atom}	atomic interchange automaton
	|	$\alpha \parallel \alpha$	parallel composition
	|	$\text{hide}_{\text{act}}(L_{\text{h}}, \alpha)$	action hiding operator
	|	$\text{hide}_{\text{var}}(X_{\text{h}}, \alpha, \sigma_{\text{h}})$	variable hiding operator
	|	$\text{urgent}_{L_{\text{u}}}(\alpha)$	urgent action operator,

where

- α_{atom} denotes an arbitrary atomic interchange automaton;
- $L_{\text{h}} \subseteq \mathcal{L}$ denotes a set of actions to hide;
- $X_{\text{h}} \subseteq \mathcal{V}$ denotes a set of variables to hide and $\sigma_{\text{h}} : X_{\text{h}} \mapsto \Lambda$ denotes a (partial) valuation for the hidden state variables of interchange automaton α;
- $L_{\text{u}} \subseteq \mathcal{L}$ denotes a set of urgent actions.

In the next sections, three auxiliary functions on interchange automata are used. These are defined below. The functions var_{e}, $\text{var}_{\text{e,state}}$ and act extract the sets of external variables, external state variables, and external (non-hidden) action labels[2], respectively, from an interchange automaton:

[2]This does not mean that these actions are actually used. It is allowed to specify the set of actions much broader than the actions that appear on transitions.

Definition 4.4. For $\alpha_a = (X, X_i, \text{dtype}, V, v, \text{init}, \text{flow}, \text{inv}, \text{urgent}, L, E)$, and $\alpha, \alpha_1, \alpha_2 \in A$ we define the functions $\text{var}_e, \text{var}_{e,\text{state}} : A \to \mathcal{P}(\mathcal{V})$ and $\text{act} : A \to \mathcal{P}(\mathcal{L})$ as follows:

$$\begin{aligned}
&\text{var}_{e,\text{state}}(\alpha_a) = X_{\text{state}} \cap X_e \\
&\text{var}_{e,\text{state}}(\alpha_1 \parallel \alpha_2) = \text{var}_{e,\text{state}}(\alpha_1) \cup \text{var}_{e,\text{state}}(\alpha_2) \\
&\text{var}_{e,\text{state}}(\text{hide}_{\text{act}}(L_h, \alpha)) = \text{var}_{e,\text{state}}(\alpha) \\
&\text{var}_{e,\text{state}}(\text{hide}_{\text{var}}(X_h, \alpha, \sigma_h)) = \text{var}_{e,\text{state}}(\alpha) \setminus X_h \\
&\text{var}_{e,\text{state}}(\text{urgent}_{L_u}(\alpha)) = \text{var}_{e,\text{state}}(\alpha)
\end{aligned}$$

$\text{var}_e(\alpha_a) = X_e$	$\text{act}(\alpha_a) = L$
$\text{var}_e(\alpha_1 \parallel \alpha_2) = \text{var}_e(\alpha_1) \cup \text{var}_e(\alpha_2)$	$\text{act}(\alpha_1 \parallel \alpha_2) = \text{act}(\alpha_1) \cup \text{act}(\alpha_2)$
$\text{var}_e(\text{hide}_{\text{act}}(L_h, \alpha)) = \text{var}_e(\alpha)$	$\text{act}(\text{hide}_{\text{act}}(L_h, \alpha)) = \text{act}(\alpha) \setminus L_h$
$\text{var}_e(\text{hide}_{\text{var}}(X_h, \alpha, \sigma_h)) = \text{var}_e(\alpha) \setminus X_h$	$\text{act}(\text{hide}_{\text{var}}(X_h, \alpha, \sigma_h)) = \text{act}(\alpha)$
$\text{var}_e(\text{urgent}_{L_u}(\alpha)) = \text{var}_e(\alpha)$	$\text{act}(\text{urgent}_{L_u}(\alpha)) = \text{act}(\alpha)$

5 Semantics of interchange automata

The formal semantics associates to each interchange automaton an action transition relation, a time transition relation, and a consistency predicate on states. A different way of looking at such a semantics is as a labeled transition system with two types of transitions and a predicate. The states S of the labeled transition system associated to an interchange automaton consist of an interchange automaton, a valuation of the external state variables of that automaton, and a set of jumping external state variables: i.e., $S = A \times \mathit{Val} \times \mathcal{P}(\mathcal{V})$, where $\mathit{Val} = \mathcal{V} \cup \dot{\mathcal{V}} \mapsto \Lambda$ is the set of all partial mappings from $\mathcal{V} \cup \dot{\mathcal{V}}$ to the set of values Λ. The set of jumping variables J is defined by other automata executing in the environment of (in parallel to) automaton α. The valuation σ of a state of a transition system defines values for precisely the externally visible state variables, i.e., $\text{dom}(\sigma) = \text{var}_{e,\text{state}}(\alpha)$ for all $(\alpha, \sigma, J) \in S$.

The intuition of an action transition $(\alpha, \sigma, J) \xrightarrow{\xi, l, W, \xi'} (\alpha', \sigma', J)$ is that the state (α, σ, J) executes a discrete action (with action label) l with visible valuations ξ, ξ', before and after execution of the action, respectively, and thereby transforms into the state (α', σ', J), where σ' denotes the accompanying valuation of the automaton α', after the discrete action l is executed. The set W represents the external state variables that are allowed to change (jump) in this action transition. They need to be visible for synchronization in a parallel composition of interchange automata.

The intuition of a time transition $(\alpha, \sigma, J) \overset{t, \rho}{\mapsto} (\alpha', \sigma', J)$ is that model time passes for t time units, and the valuation at each time-point $s \in [0, t]$ is given by $\rho(s)$ for the externally visible variables. At the end-point t, the resulting state is (α', σ', J).

The intuition of the consistency predicate $(\alpha, \sigma, J) \overset{\xi}{\rightsquigarrow}$ is that the interchange automaton α is consistent with extended valuation ξ, which means that the invariants of all active locations of α are satisfied in ξ.

Notation 5.1. In this section, some notations and operators are used. These are defined as follows:

- $\restriction$ is the restriction operator on functions. If f is a function, and S is a set, $f \restriction S$ denotes the restriction of f to S, that is, the function g with $\text{dom}(g) = \text{dom}(f) \cap S$, such that $g(c) = f(c)$ for each $c \in \text{dom}(g)$.
- $\downarrow$ is the projection operator on functions, which is used here on trajectories. For $\rho : T \mapsto (Y \to \Lambda)$, $S \subseteq Y$ and $x \in Y$, $\rho \downarrow S$ denotes the function $\rho' : T \mapsto (S \to \Lambda)$ such that $\rho'(t) = \rho(t) \restriction S$ for each $t \in T$; and $\rho \downarrow x$ denotes the function $f : T \mapsto \Lambda$ such that $f(t) = \rho(t)(x)$ for each $t \in \text{dom}(\rho)$.
- For atomic interchange automaton α_{atom} given by $(X, X_i, \text{dtype}, V, v, \text{init}, \text{flow}, \text{inv}, \text{urgent}, L, E)$, $\alpha_{\text{atom}}[v', \text{init}'/v, \text{init}]$ denotes the atomic interchange automaton obtained

from atomic interchange automaton α_{atom} by replacing v by v' and init by init$'$.

5.1 Semantics of atomic interchange automata

Definition 5.2 (Action transitions)**.** Consider an atomic interchange automaton $\alpha = (X,\ X_{\text{i}},$ dtype, $V,\ v,$ init, flow, inv, urgent, $L,\ E)$. The *action transition relation* $\_ \xrightarrow{} \_ \subseteq S \times (\mathit{Val} \times \mathcal{L}_\tau \times \mathcal{P}(\mathcal{V}) \times \mathit{Val}) \times S$ is for $(\alpha, \sigma, J), (\alpha', \sigma', J) \in S, \xi_{\text{e}}, \xi'_{\text{e}} \in \mathit{Val}, l \in \mathcal{L}_\tau$, and $W_{\text{e}} \subseteq \mathcal{V}$, defined as follows: $(\alpha, \sigma, J) \xrightarrow{\xi_{\text{e}}, l, W_{\text{e}}, \xi'_{\text{e}}} (\alpha', \sigma', J)$, if and only if there exist and edge $(v, g, l, (W, r), v') \in E$ and $\xi, \xi' \in \mathit{Val}$ with $\text{dom}(\xi) = \text{dom}(\xi') = \tilde{X}$ such that

- $\xi \upharpoonright \tilde{X_{\text{e}}} = \xi_{\text{e}}$ and $\xi' \upharpoonright \tilde{X_{\text{e}}} = \xi'_{\text{e}}$;
- $\xi_{\text{e}} \upharpoonright X_{\text{state}} = \sigma$ and $\xi'_{\text{e}} \upharpoonright X_{\text{state}} = \sigma'$;
- $\xi \models \text{init}$ and $\xi \models g$;
- $\xi \models \text{inv}(v)$ and $\xi' \models \text{inv}(v')$;
- $\xi' \cup \xi^- \models r$;
- $\xi \upharpoonright X_{\text{nonjmp}} = \xi' \upharpoonright X_{\text{nonjmp}}$, where $X_{\text{nonjmp}} = X_{\text{state}} \setminus (W \cup (J \cap X_{\text{e}}))$;
- $W_{\text{e}} = W \cap X_{\text{state}} \cap X_{\text{e}}$;
- $\alpha' = \alpha[v', \text{init}'/v, \text{init}]$, $\text{init}' = \left(\bigwedge_{x \in X_{\text{state}} \cap X_{\text{i}}} x = c_x \right)$, and $c_x \in \Lambda$ is given by $c_x = \xi'(x)$.

Minus superscripted variables, such as x^-, occurring in r are evaluated in ξ^-, which is defined as $\text{dom}(\xi^-) = \{x^- \mid x \in \text{dom}(\xi)\}$, and $\xi^-(x^-) = \xi(x)$. The 'non-jumping' variables in the set $X_{\text{nonjmp}} = X_{\text{state}} \setminus (W \cup (J \cap X_{\text{e}}))$ are the variables the values of which are not allowed to change in an action transition. These variables are the discrete and continuous variables apart from two sets of variables: the variables from set W and the externally visible variables from set J $(J \cap X_{\text{e}})$. The jumping variables in set J are the result of changes in external variables of synchronizing automata, as defined in the semantics of parallel composition Rule 1. The updated initial condition init$'$ acts as a local valuation which ensures that for each local state variable x, its starting value for the next transition equals its resulting value (here: $\xi'(x)$, in Definition 5.3: $\rho(t)(x)$) for the current transition.

Definition 5.3 (Time transitions)**.** Consider an atomic interchange automaton $\alpha = (X, X_{\text{i}},$ dtype, $V,\ v,$ init, flow, inv, urgent, $L,\ E)$. The *time transition relation* $\_ \xrightarrow{} \_ \subseteq S \times (T \times (T \mapsto \mathit{Val})) \times S$ is for $(\alpha, \sigma, J), (\alpha', \sigma', J) \in S,\ t \in T$, and $\rho_{\text{e}} : [0, t] \to \mathit{Val}$, defined as follows: $(\alpha, \sigma, J) \overset{t, \rho_{\text{e}}}{\mapsto} (\alpha', \sigma', J)$, if and only if there exists a $\rho : [0, t] \to \mathit{Val}$ with $\text{dom}(\rho(s)) = \tilde{X}$ for all $s \in [0, t]$ such that

- $\rho \downarrow \tilde{X_{\text{e}}} = \rho_{\text{e}}$;
- $\rho_{\text{e}}(0) \upharpoonright X_{\text{state}} = \sigma$ and $\rho_{\text{e}}(t) \upharpoonright X_{\text{state}} = \sigma'$;
- $\rho(0) \models \text{init}$;
- $\rho \downarrow x$ is a constant function for all $x \in X_{\text{disc}}$;
- $(\rho \downarrow x) \in F$ for all $x \in X_{\text{alg}}$;
- $\rho \downarrow \dot{x}$ is an integrable function in the Lebesgue sense for all $x \in X_{\text{cont}}$;
- $\rho(s) \models \text{flow}(v)$ and $\rho(s) \models \text{inv}(v)$ for all $s \in [0, t]$;

- $(\rho \downarrow x)(s) = (\rho \downarrow x)(0) + \int_0^s (\rho \downarrow \dot{x})(s')ds'$ for all $x \in X_{\text{cont}}$ and $s \in [0, t]$;
- $(\rho \downarrow x, \rho \downarrow \dot{x}) \in G$ for all $x \in X_{\text{cont}}$;
- $\rho(s) \models \neg\text{urgent}(v)$ for all $s \in \{0\} \cup [0, t)$;
- $\alpha' = \alpha[\text{init}'/\text{init}]$, $\text{init}' = \left(\bigwedge_{x \in X_{\text{state}} \cap X_{\text{i}}} x = c_x \right)$, and $c_x \in \Lambda$ is given by $c_x = \rho(t)(x)$.

Item $(\rho \downarrow x) \in F$ for all $x \in X_{\text{alg}}$, requires the trajectories of the algebraic variables to be functions of type F. This set of functions is a global parameter of the solution concept of an interchange automaton specification.

The trajectories of the dotted variables are required to be integrable. This ensures that the integral $\int_0^s (\rho \downarrow \dot{x})(s')ds'$ is defined. The relation between the trajectory of a continuous variable x and the trajectory of its 'derivative' $\dot{x}$ is given by the Caratheodory solution concept [12]: $(\rho \downarrow x)(s) = (\rho \downarrow x)(0) + \int_0^s (\rho \downarrow \dot{x})(s')ds'$. This integral relation can hold only for those continuous variables for which $\rho \downarrow x$ is an absolutely continuous function, but it does allow a non-smooth trajectory for a continuous variable in the case that the trajectory of its 'derivative' $\rho \downarrow \dot{x}$ is non-smooth or even discontinuous, as in, for example, in the solution of the invariant $\dot{y} = \text{step}(t - 1) \wedge \dot{t} = 1$, where t and y are continuous variable with initial value of 0, and $\text{step}(x)$ equals 0 for $x \leq 0$ and 1 for $x > 0$.

In hybrid automata, the solution concept usually defines the function $\rho \downarrow \dot{x}$ to be the derivative function of $\rho \downarrow x$ for continuous variables $x \in X_{\text{cont}}$. This can be realized for the interchange automaton format semantics by restricting the set G, which is used in the requirement $(\rho \downarrow x, \rho \downarrow \dot{x}) \in G$ for all $x \in X_{\text{cont}}$, as $G = \{(f, f') \mid f \text{ is differentiable, and } f' \text{ is the derivative function of } f\}$. In this way, the semantics of the interchange automaton format corresponds to the usual semantics of hybrid automata. Piecewise continuous functions for the trajectories of the algebraic and dotted variables can be expressed by means of: $F = \{f \mid f \text{ is a piecewise continuous function}\}$, $G = \{(f, f') \mid f' \text{ is a piecewise continuous function}\}$. Another possibility would be not to define additional restrictions: $F = \{f \mid \text{true}\}$, $G = \{(f, f') \mid \text{true}\}$. For an invariant such as $\dot{x} = y,\ y = \text{step}(t - 1),\ \dot{t} = 1$, which has just one solution for continuous variables x, t and algebraic variable y, the solution is the same for both cases of F and G. For an invariant 'true' that allows infinitely many solutions, there would obviously be a difference.

Definition 5.4 (Consistency predicates)**.** Consider an atomic interchange automaton $\alpha = (X, X_{\text{i}}, \text{dtype}, V, v, \text{init}, \text{flow}, \text{inv}, \text{urgent}, L, E)$. The *consistency predicate* $\_ \overset{\sim}{\rightarrow} \subseteq S \times \mathit{Val}$ is for $(\alpha, \sigma, J) \in S$ and $\xi_{\text{e}} \in \mathit{Val}$, defined as follows: $(\alpha, \sigma, J) \overset{\xi_{\text{e}}}{\overset{\sim}{\rightarrow}}$, if and only if there exists a valuation $\xi \in \mathit{Val}$ with $\text{dom}(\xi) = \tilde{X}$ such that $\xi \restriction \tilde{X}_{\text{e}} = \xi_{\text{e}}$, $\xi_{\text{e}} \restriction X_{\text{state}} = \sigma$, $\xi \models \text{init}$, and $\xi \models \text{inv}(v)$.

5.2 Semantics of the operators

The formal semantics of the operators is defined in a structured operational semantics (SOS) style [31] below.

Notation 5.5. The following notations are used in this section:

- For functions $f, g : A \mapsto B$, we define $f \doteq g$ iff $f(a) = g(a)$ for all $a \in \text{dom}(f) \cap \text{dom}(g)$. Similarly, for functions $f, g : T \mapsto A \mapsto B$ with $\text{dom}(f) = \text{dom}(g)$, we define $f \doteqdot g$ iff $f(t) \doteq g(t)$ for all $t \in \text{dom}(f)$.
- If f and g are functions with $f \doteq g$, then $f \cup g$ denotes the function h with $\text{dom}(h) = \text{dom}(f) \cup \text{dom}(g)$ satisfying the condition: for each $c \in \text{dom}(h)$, if $c \in \text{dom}(f)$ then $h(c) = f(c)$, and $h(c) = g(c)$ otherwise. Similarly, if f and g are functions with $f \doteqdot g$, then $f \Cup g$ denotes the function h with $\text{dom}(h) = \text{dom}(f)$ satisfying the condition: for each $t \in \text{dom}(h)$, $h(t) = f(t) \cup g(t)$.

- For functions $f, g : A \mapsto B$, we define $f \succ g$ to denote the function h with $\mathrm{dom}(h) = \mathrm{dom}(f) \cup \mathrm{dom}(g)$ satisfying the condition: for each $a \in \mathrm{dom}(h)$, if $a \in \mathrm{dom}(f)$ then $h(a) = f(a)$, and $h(a) = g(a)$ otherwise.
- The notation $\frac{H}{R}$, where R is a number of results separated by commas, as used in Rule 2, is an abbreviation for a set of deduction rules of the form $\frac{H}{r}$; one for each $r \in R$.

Parallel composition

The most common operator for composing hybrid automata is parallel composition. There are no compatibility requirements for the parallel composition of interchange automata: any pair of interchange automata can be composed by the parallel composition operator. The parallel composition operator synchronizes on all external actions that the arguments share and allows interleaving of any other actions (under the condition that they maintain the consistency of the other automaton). Time transitions must be synchronized, and consistency is established only if both automata agree on it. The external state variables that are shared by the argument automata need to have the same values (all the time).

$$\frac{(\alpha_1, \sigma_{\alpha_1}, J \cup W_2) \xrightarrow{\xi_1, l, W_1, \xi_1'} (\alpha_1', \sigma_1', J'), (\alpha_2, \sigma_{\alpha_2}, J \cup W_1) \xrightarrow{\xi_2, l, W_2, \xi_2'} (\alpha_2', \sigma_2', J''), \quad \xi_1 \doteq \xi_2,\ \xi_1' \doteq \xi_2',\ l \neq \tau}{(\alpha_1 \parallel \alpha_2, \sigma, J) \xrightarrow{\xi_1 \cup \xi_2, l, W_1 \cup W_2, \xi_1' \cup \xi_2'} (\alpha_1' \parallel \alpha_2', \sigma_1' \cup \sigma_2', J)}\ 1$$

$$\frac{\begin{array}{c}(\alpha_2, \sigma_{\alpha_2}, J) \overset{\xi_2}{\rightsquigarrow},\ (\alpha_1, \sigma_{\alpha_1}, J) \xrightarrow{\xi_1, l, W, \xi_1'} (\alpha_1', \sigma_1', J'), \\ (\alpha_2, \sigma_1' \restriction \mathrm{dom}(\sigma_{\alpha_2}) \succ \sigma_{\alpha_2}, J) \overset{\xi_2'}{\rightsquigarrow},\ \xi_1 \doteq \xi_2,\ \xi_1' \doteq \xi_2',\ l \notin \mathrm{act}(\alpha_2)\end{array}}{\begin{array}{c}(\alpha_1 \parallel \alpha_2, \sigma, J) \xrightarrow{\xi_1 \cup \xi_2, l, W, \xi_1' \cup \xi_2'} (\alpha_1' \parallel \alpha_2, \sigma_1' \succ \sigma_2', J), \\ (\alpha_2 \parallel \alpha_1, \sigma, J) \xrightarrow{\xi_2 \cup \xi_1, l, W, \xi_2' \cup \xi_1'} (\alpha_2 \parallel \alpha_1', \sigma_1' \succ \sigma_2', J)\end{array}}\ 2$$

$$\frac{(\alpha_1, \sigma_{\alpha_1}, J) \overset{t, \rho_1}{\longmapsto} (\alpha_1', \sigma_1', J'),\ (\alpha_2, \sigma_{\alpha_2}, J) \overset{t, \rho_2}{\longmapsto} (\alpha_2', \sigma_2', J''),\ \rho_1 \fallingdotseq \rho_2}{(\alpha_1 \parallel \alpha_2, \sigma, J) \overset{t, \rho_1 \Cup \rho_2}{\longmapsto} (\alpha_1' \parallel \alpha_2', \sigma_1' \cup \sigma_2', J)}\ 3$$

$$\frac{(\alpha_1, \sigma_{\alpha_1}, J) \overset{\xi_1}{\rightsquigarrow},\ (\alpha_2, \sigma_{\alpha_2}, J) \overset{\xi_2}{\rightsquigarrow},\ \xi_1 \doteq \xi_2}{(\alpha_1 \parallel \alpha_2, \sigma, J) \overset{\xi_1 \cup \xi_2}{\rightsquigarrow}}\ 4$$

Here, σ_{α_1} and σ_{α_2} are abbreviations for $\sigma \restriction \mathrm{var}_{\mathrm{e,state}}(\alpha_1)$ and $\sigma \restriction \mathrm{var}_{\mathrm{e,state}}(\alpha_2)$, respectively. Note that by definition (see the definition of the state S in Section 5), $\mathrm{dom}(\sigma) = \mathrm{var}_{\mathrm{e,state}}(\alpha_1 \parallel \alpha_2)$. The valuation $\sigma_1' \restriction \mathrm{dom}(\sigma_{\alpha_2}) \succ \sigma_{\alpha_2}$ in Rule 2 represents the updated version of valuation σ_{α_2} such that the valuations for the shared variables ($\mathrm{var}_{\mathrm{e,state}}(\alpha_1) \cap \mathrm{var}_{\mathrm{e,state}}(\alpha_2)$) in σ_{α_2} have been overwritten by the values defined in σ_1'. In this way, Rule 2 allows an automaton to change the values of variables shared with another automaton, as long as the new valuations for the shared variables are consistent with the (invariants of the) other automaton.

Action hiding

The action hiding operator applied to an automaton, $\mathrm{hide}_{\mathrm{act}}(L_{\mathrm{h}}, \alpha)$, hides (abstracts from) the actions from set L_{h} by replacing them by the internal action τ. This only affects the action behavior of α; its delay behavior and consistency remain unchanged.

$$\frac{(\alpha, \sigma, J) \xrightarrow{\xi, l, W, \xi'} (\alpha', \sigma', J'),\ l \notin L_{\mathrm{h}}}{(\mathrm{hide}_{\mathrm{act}}(L_{\mathrm{h}}, \alpha), \sigma, J) \xrightarrow{\xi, l, W, \xi'} (\mathrm{hide}_{\mathrm{act}}(L_{\mathrm{h}}, \alpha'), \sigma', J)}\ 5$$

$$\frac{(\alpha,\sigma,J)\xrightarrow{\xi,l,W,\xi'}(\alpha',\sigma',J'),\ l\in L_{\mathrm{h}}}{(\mathrm{hide}_{\mathrm{act}}(L_{\mathrm{h}},\alpha),\sigma,J)\xrightarrow{\xi,\tau,W,\xi'}(\mathrm{hide}_{\mathrm{act}}(L_{\mathrm{h}},\alpha'),\sigma',J)}\ 6$$

$$\frac{(\alpha,\sigma,J)\overset{t,\rho}{\longmapsto}(\alpha',\sigma',J')}{(\mathrm{hide}_{\mathrm{act}}(L_{\mathrm{h}},\alpha),\sigma,J)\overset{t,\rho}{\longmapsto}(\mathrm{hide}_{\mathrm{act}}(L_{\mathrm{h}},\alpha'),\sigma',J)}\ 7 \qquad \frac{(\alpha,\sigma,J)\overset{\xi}{\rightsquigarrow}}{(\mathrm{hide}_{\mathrm{act}}(L_{\mathrm{h}},\alpha),\sigma,J)\overset{\xi}{\rightsquigarrow}}\ 8,$$

Variable hiding

The variable hiding operator applied to an automaton, $\mathrm{hide}_{\mathrm{var}}(X_{\mathrm{h}},\alpha,\sigma_{\mathrm{h}})$, hides the variables from set X_{h} by removing information about them from the action and time transitions of α. The values of the hidden state variables are stored in valuation σ_{h}.

$$\frac{(\alpha,\sigma\cup\sigma'_{\mathrm{h}},J\setminus X_{\mathrm{h}})\xrightarrow{\xi,l,W,\xi'}(\alpha',\sigma',J'),\ \sigma'_{\mathrm{h}}\mathrel{\widehat{=}}\sigma_{\mathrm{h}},\ \mathrm{dom}(\sigma'_{\mathrm{h}})=\mathrm{var}_{\mathrm{e,state}}(\alpha)\cap X_{\mathrm{h}}}{(\mathrm{hide}_{\mathrm{var}}(X_{\mathrm{h}},\alpha,\sigma_{\mathrm{h}}),\sigma,J)\xrightarrow{\xi_{\mathrm{vis}},l,W_{\mathrm{vis}},\xi'_{\mathrm{vis}}}(\mathrm{hide}_{\mathrm{var}}(X_{\mathrm{h}},\alpha',\sigma'_{\sigma'_{\mathrm{h}}}),\sigma'_{\sigma},J)}\ 9$$

$$\frac{(\alpha,\sigma\cup\sigma'_{\mathrm{h}},J\setminus X_{\mathrm{h}})\overset{t,\rho}{\longmapsto}(\alpha',\sigma',J'),\ \sigma'_{\mathrm{h}}\mathrel{\widehat{=}}\sigma_{\mathrm{h}},\ \mathrm{dom}(\sigma'_{\mathrm{h}})=\mathrm{var}_{\mathrm{e,state}}(\alpha)\cap X_{\mathrm{h}}}{(\mathrm{hide}_{\mathrm{var}}(X_{\mathrm{h}},\alpha,\sigma_{\mathrm{h}}),\sigma,J)\overset{t,\rho_{\mathrm{vis}}}{\longmapsto}(\mathrm{hide}_{\mathrm{var}}(X_{\mathrm{h}},\alpha',\sigma'_{\sigma'_{\mathrm{h}}}),\sigma'_{\sigma},J)}\ 10$$

$$\frac{(\alpha,\sigma\cup\sigma'_{\mathrm{h}},J\setminus X_{\mathrm{h}})\overset{\xi}{\rightsquigarrow},\ \sigma'_{\mathrm{h}}\mathrel{\widehat{=}}\sigma_{\mathrm{h}},\ \mathrm{dom}(\sigma'_{\mathrm{h}})=\mathrm{var}_{\mathrm{e,state}}(\alpha)\cap X_{\mathrm{h}}}{(\mathrm{hide}_{\mathrm{var}}(X_{\mathrm{h}},\alpha,\sigma_{\mathrm{h}}),\sigma,J)\overset{\xi_{\mathrm{vis}}}{\rightsquigarrow}}\ 11$$

Notations $\sigma'_{\sigma'_{\mathrm{h}}}$ and σ'_{σ} are abbreviations for $\sigma'\restriction\mathrm{dom}(\sigma'_{\mathrm{h}})$ and $\sigma'\restriction\mathrm{dom}(\sigma)$, respectively. The visible valuations ξ_{vis} and ξ'_{vis}, visible set of jumping variables W_{vis}, and visible trajectory ρ_{vis}, are abbreviations for $\xi\restriction(X_{\mathrm{vis}}\cup\dot{X}_{\mathrm{vis}})$, $\xi'\restriction(X_{\mathrm{vis}}\cup\dot{X}_{\mathrm{vis}})$, $W\restriction X_{\mathrm{vis}}$, and $\rho\downarrow(X_{\mathrm{vis}}\cup\dot{X}_{\mathrm{vis}})$, respectively, with $X_{\mathrm{vis}}=\mathrm{var}_{\mathrm{e}}(\alpha)\setminus X_{\mathrm{h}}$.

Valuation $\sigma_{\mathrm{h}}:X_{\mathrm{h}}\mapsto\Lambda$ is a partial function that defines the initial values for (some of) the hidden variables. These initial values are relevant only for the hidden state variables: valuation $\sigma'_{\mathrm{h}}:\mathrm{var}_{\mathrm{e,state}}(\alpha)\cap X_{\mathrm{h}}\to\Lambda$ defines the initial values for the hidden state variables and takes them from σ_{h} if possible and takes arbitrary values otherwise: $\sigma'_{\mathrm{h}}\mathrel{\widehat{=}}\sigma_{\mathrm{h}}$.

Urgent action operator

The urgent action operator applied to an automaton, $\mathrm{urgent}_{L_{\mathrm{u}}}(\alpha)$, gives actions from the set L_{u} priority over time passing. The action behavior and consistency of α are not affected by the urgent action operator. Time transitions are allowed only if at the current state, and at each intermediate state while delaying, no actions from set L_{u} are possible.

$$\frac{(\alpha,\sigma,J)\xrightarrow{\xi,l,W,\xi'}(\alpha',\sigma',J')}{(\mathrm{urgent}_{L_{\mathrm{u}}}(\alpha),\sigma,J)\xrightarrow{\xi,l,W,\xi'}(\mathrm{urgent}_{L_{\mathrm{u}}}(\alpha'),\sigma',J)}\ 12 \qquad \frac{(\alpha,\sigma,J)\overset{\xi}{\rightsquigarrow}}{(\mathrm{urgent}_{L_{\mathrm{u}}}(\alpha),\sigma,J)\overset{\xi}{\rightsquigarrow}}\ 13$$

$$\frac{\begin{array}{l}(\alpha,\sigma,J)\overset{t,\rho}{\longmapsto}(\alpha',\sigma',J'),\ \forall_{l\in L_{\mathrm{u}}}(\alpha,\sigma,J)\overset{l}{\not\mapsto},\\ \forall_{s\in[0,t)}\,(\exists_{\alpha'',\sigma'',J''}\quad(\alpha,\sigma,J)\overset{s,\rho\restriction[0,s]}{\longmapsto}(\alpha'',\sigma'',J''),\\ \qquad\qquad(\alpha'',\sigma'',J'')\overset{t-s,\rho_{-s}}{\longmapsto}(\alpha',\sigma',J'),\ \forall_{l\in L_{\mathrm{u}}}(\alpha'',\sigma'',J'')\overset{l}{\not\mapsto})\end{array}}{(\mathrm{urgent}_{L_{\mathrm{u}}}(\alpha),\sigma,J)\overset{t,\rho}{\longmapsto}(\mathrm{urgent}_{L_{\mathrm{u}}}(\alpha'),\sigma',J)}\ 14$$

Here ρ_{-s} denotes the trajectory ρ shifted left by s time-units and starting at 0: $\mathrm{dom}(\rho_{-s})=[0,t-s]$, assuming $\mathrm{dom}(\rho)=[0,t]$, and $\forall_{t'\in\mathrm{dom}(\rho_{-s})}\,\rho_{-s}(t')=\rho(t'+s)$. Furthermore, notation $(\alpha,\sigma,J)\overset{l}{\not\mapsto}$ denotes $\neg\exists_{\xi,W,\xi',\alpha',\sigma',J'}\,(\alpha,\sigma,J)\xrightarrow{\xi,l,W,\xi'}(\alpha',\sigma',J')$.

5.3 Equality and compositionality

Two interchange automata α_1 and α_2 are considered *comparable* iff they have the same sets of externally visible state variables and actions: i.e., $\text{var}_{\text{e,state}}(\alpha_1) = \text{var}_{\text{e,state}}(\alpha_2)$ and $\text{act}(\alpha_1) = \text{act}(\alpha_2)$. Next, we define when two comparable interchange automata are equivalent. For this we use the notion of stateless bisimilarity as defined in [28].

Definition 5.6. A symmetric relation R on comparable interchange automata is a (stateless) bisimulation relation, if and only if for all $(\alpha_1, \alpha_2) \in R$ and for all $\alpha_1', \sigma, \sigma', J, J', \xi, \xi', l, W, t, \rho$:

- if $(\alpha_1, \sigma, J) \xrightarrow{\xi,l,W,\xi'} (\alpha_1', \sigma', J')$, then $(\alpha_2, \sigma, J) \xrightarrow{\xi,l,W,\xi'} (\alpha_2', \sigma', J')$ for some α_2' such that $(\alpha_1', \alpha_2') \in R$;
- if $(\alpha_1, \sigma, J) \overset{t,\rho}{\longmapsto} (\alpha_1', \sigma', J')$, then $(\alpha_2, \sigma, J) \overset{t,\rho}{\longmapsto} (\alpha_2', \sigma', J')$ for some α_2' such that $(\alpha_1', \alpha_2') \in R$;
- if $(\alpha_1, \sigma, J) \overset{\xi}{\rightsquigarrow}$, then $(\alpha_2, \sigma, J) \overset{\xi}{\rightsquigarrow}$.

Two interchange automata α_1 and α_2 are (stateless) *bisimilar*, notation $\alpha_1 \underline{\leftrightarrow} \alpha_2$, if and only if there exists a (stateless) bisimulation relation R such that $(\alpha_1, \alpha_2) \in R$.

Note that if an automaton α_1 is comparable with another automaton α_2, then any automaton α_1' that can be reached from automaton α_1 by performing action and time transitions is also comparable with α_2. Therefore, the above restriction on a bisimulation relation that only comparable automata can be related needs to be verified/established only for the automata that one wishes to compare.

Theorem 5.7. *Bisimilarity is a congruence for all operators introduced in this article.*

Proof. Apart from Rule 14 for the urgent action operator, it is easy to see that all deduction rules of the operators of the interchange automaton format satisfy the *process-tyft* format containing predicates [28] (which we call process-path format for simplicity). Thus for all operators, except for the urgent action operator, it can be concluded that stateless bisimilarity is a congruence.

Rule 14 does not satisfy the process-path format. Therefore we need to give manual proof to show that stateless bisimilarity is a congruence for the urgent action operator. The proof required is similar to the proof given in [25], pages 160–161. From this proof, we can conclude that stateless bisimilarity is also a congruence for the urgent action operator. □

6 Elimination of the operators

This section defines how an interchange automaton specification can be rewritten as a single atomic interchange automaton that is bisimilar to the original specification. For this purpose, we define how the operators of the interchange automaton language can be eliminated when they are applied to one or more atomic interchange automata.

6.1 Elimination of parallel composition

Let α_1 and α_2 denote two atomic interchange automata the shared variables of which have compatible types (defined below), and let two renaming functions R_1 and R_2 be defined on atomic interchange automata, in such a way, that the internal variables of the renamed automata $R_1(\alpha_1)$ and $R_2(\alpha_2)$ become unique (different from all other variables). The parallel composition operator can then be eliminated as defined below.

Let $R_1(\alpha_1) = (X_1, X_{\text{i}1}, \text{dtype}_1, V_1, v_{0_1}, \text{init}_1, \text{flow}_1, \text{inv}_1, \text{urgent}_1, L_1, E_1)$ and $R_2(\alpha_2) = (X_2, X_{\text{i}2}, \text{dtype}_2, V_2, v_{0_2}, \text{init}_2, \text{flow}_2, \text{inv}_2, \text{urgent}_2, L_2, E_2)$ denote the two atomic interchange automata that are obtained by renaming the internal variables of α_1 and α_2, so that $X_{\text{i}1} \cap X_2 = \emptyset$ and

$X_{i2} \cap X_1 = \emptyset$. Then the parallel composition $\alpha_1 \parallel \alpha_2$ can be rewritten as an atomic interchange automaton $\alpha = (X, X_i, \text{dtype}, V, v_0, \text{init}, \text{flow}, \text{inv}, \text{urgent}, L, E)$ such that:

- $X = X_1 \cup X_2$;
- $X_i = X_{i1} \cup X_{i2}$;
- for all $x \in X$: $\text{dtype}(x) = \begin{cases} \text{dtype}_1(x) & \text{if } x \in X_1 \setminus X_2 \\ \text{dtype}_2(x) & \text{if } x \in X_2 \setminus X_1 \\ \text{mdt}(\text{dtype}_1(x), \text{dtype}_2(x)) & \text{if } x \in X_1 \cap X_2 \end{cases}$
- $V = V_1 \times V_2$;
- $v_0 = (v_{0_1}, v_{0_2})$;
- $\text{init} = \text{init}_1 \wedge \text{init}_2$;
- $\text{flow}(v_1, v_2) = \text{flow}_1(v_1) \wedge \text{flow}_2(v_2)$, $\text{inv}(v_1, v_2) = \text{inv}_1(v_1) \wedge \text{inv}_2(v_2)$, $\text{urgent}(v_1, v_2) = \text{urgent}_1(v_1) \vee \text{urgent}_2(v_2)$;
- $L = L_1 \cup L_2$;
- E contains the edge $((v_1, v_2), g, l, (W, r), (v_1', v_2'))$ if and only if:
  - there is an edge $(v_2, g, l, (W, r), v_2') \in E_2$ such that $l \notin L_1$, and $v_1 = v_1'$;
  - there is an edge $(v_1, g, l, (W, r), v_1') \in E_1$ such that $l \notin L_2$, and $v_2 = v_2'$;
  - there is an edge $(v_1, g_1, l, (W_1, r_1), v_1') \in E_1$ and an edge $(v_2, g_2, l, (W_2, r_2), v_2') \in E_2$ such that $l \neq \tau$, and $W = W_1 \cup W_2$, $r = r_1 \wedge r_2$, $g = g_1 \wedge g_2$.

The function $\text{mdt} \in \mathcal{D} \to \mathcal{D}$ merges the dynamic types from set $\mathcal{D} = \{\text{disc}, \text{cont}, \text{alg}\}$ as follows: $\text{mdt}(\text{disc}, \text{disc}) = \text{disc}$, $\text{mdt}(\text{cont}, \text{cont}) = \text{cont}$, and $\text{mdt}(\text{alg}, d) = \text{mdt}(d, \text{alg}) = d$ for all $d \in \mathcal{D}$. In principle, we could define $\text{mdt}(\text{cont}, \text{disc}) = \text{mdt}(\text{disc}, \text{cont}) = \text{cont}$ and define an additional equation $\dot{x} = 0$ for all variables that are of dynamic type disc in one of the parallel automata and of dynamic type cont in the other automaton. This is not done in order to keep the elimination of parallel composition straightforward. Therefore, compatible types are defined as follows: the dynamic type alg is compatible with all other dynamic types and identical types are compatible. We conjecture that the automata $\alpha_1 \parallel \alpha_2$ and α are bisimilar.

Renaming the internal variables is needed in general, because elimination of the parallel composition operator merges the variables of the two parallel automata, and only the external variables with the same names should be shared. This renaming of internal variables does not change the meaning of the automata, because the internal variables do not appear on the transition system (see Section 5). Note that elimination of the parallel composition operator does not require actions to be renamed, because the actions of sets L, L_1, L_2 are all external. Hidden actions are renamed to the non-synchronizing τ action label, either by the action hiding operator, or by elimination (see Sections 5.2 and 6.2, respectively).

6.2 Elimination of action hiding

Let $\alpha = (X, X_i, \text{dtype}, V, v_0, \text{init}, \text{flow}, \text{inv}, \text{urgent}, L, E)$ denote an atomic interchange automaton. The action hiding operator applied to this atomic interchange automaton $\text{hide}_{\text{act}}(L_h, \alpha)$ can be rewritten as an atomic interchange automaton $\alpha' = (X, X_i, \text{dtype}, V, v_0, \text{init}, \text{flow}, \text{inv}, \text{urgent}, L', E')$ such that:

- $L' = L \setminus L_h$;
- $E' = \{(v, g, l, (W, r), v') \mid (v, g, l, (W, r), v') \in E, l \notin L_h\} \cup \{(v, g, \tau, (W, r), v') \mid (v, g, l, (W, r), v') \in E, l \in L_h\}$.

We conjecture that the automata $\text{hide}_{\text{act}}(L_h, \alpha)$ and α' are bisimilar.

6.3 Elimination of variable hiding

Let $\alpha = (X,\ X_\mathrm{i},\ \text{dtype},\ V,\ v_0,\ \text{init},\ \text{flow},\ \text{inv},\ \text{urgent},\ L,\ E)$ denote an atomic interchange automaton. The action hiding applied to this atomic interchange automaton $\text{hide}_\text{var}(X_\mathrm{h},\ \alpha,\ \sigma_\mathrm{h})$ can be rewritten as an atomic interchange automaton $\alpha' = (X,\ X_\mathrm{i}',\ \text{dtype},\ V,\ v_0,\ \text{init}',\ \text{flow},\ \text{inv},\ \text{urgent},\ L,\ E)$ such that:

- $X_\mathrm{i}' = X_\mathrm{i} \cup (X_\mathrm{h} \cap X_\mathrm{e})$;
- $\text{init}' = \text{init} \wedge \left(\bigwedge_{x \in \text{dom}(\sigma_\mathrm{h}) \cap X_\mathrm{e} \cap X_\text{state}} x = c_x \right)$, where $c_x \in \Lambda$ is given by $c_x = \sigma_\mathrm{h}(x)$.

We conjecture that the automata $\text{hide}_\text{var}(X_\mathrm{h},\ \alpha,\ \sigma_\mathrm{h})$ and α' are bisimilar.

6.4 Elimination of the urgent action operator

For a subclass of atomic interchange automata characterized as being 'reactive' (defined below) for all urgent actions, the urgent action operator can be eliminated by redefining the urgency conditions. Let $\alpha = (X,\ X_\mathrm{i},\ \text{dtype},\ V,\ v_0,\ \text{init},\ \text{flow},\ \text{inv},\ \text{urgent},\ L,\ E)$ denote an atomic interchange automaton, let $L_\mathrm{u} \subseteq \mathcal{L}$ be a set of urgent actions, and let automaton α be reactive for all actions of set L_u. Then the result of applying the urgent action operator with urgent action set L_u on α, denoted as $\text{urgent}_{L_\mathrm{u}}(\alpha)$, can be rewritten as the interchange automaton $\alpha' = (X,\ X_\mathrm{i},\ \text{dtype},\ V,\ v_0,\ \text{init},\ \text{flow},\ \text{inv},\ \text{urgent}',\ L,\ E)$, such that the urgency condition of each location in α' is the disjunction of 1) the urgency condition of the location in α, and 2) the guards of the outgoing urgent edges of the location, where an urgent edge is an edge with an action $l \in L_\mathrm{u}$:

$$\text{urgent}'(v) = \text{urgent}(v)\ \vee \left(\bigvee_{g \in \text{ug}(v)} g \right),\ \text{where ug}(v) = \{g \mid (v, g, l, (W, r), v') \in E,\ l \in L_\mathrm{u}\}.$$

We conjecture that the automata $\text{urgent}_{L_\mathrm{u}}(\alpha)$ and α' are bisimilar.

An atomic interchange automaton is reactive for a set of actions L_u if whenever the guard of an edge labeled with an urgent action label $l \in L_\mathrm{u}$ can be satisfied, the automaton has an action transition for action l. This means that the invariants of the target locations of edges with urgent actions (and the jump predicates of the edges) should not prevent the action from taking place. Formally, an atomic interchange automaton $(X,\ X_\mathrm{i},\ \text{dtype},\ V,\ v_0,\ \text{init},\ \text{flow},\ \text{inv},\ \text{urgent},\ L,\ E)$ is reactive for a set of actions $L_\mathrm{u} \subseteq L$ if

$$\forall_{(v,\xi,J)\in\hat{S}}\ (\forall_{(v,g,l,(W,r),v')\in E_\mathrm{u}}\ \xi \models g \implies (\alpha_v, \xi \restriction X_\text{state},\ J) \xrightarrow{\xi_\mathrm{e}, l, W, \xi_\mathrm{e}'} (\alpha', \sigma,\ J)),$$

for some ξ_e, W, ξ_e', α', and σ, where the set of admissible extended states $\hat{S}$ is defined as: $\hat{S} = \{(v,\ \xi,\ J) \in V \times \mathit{Val} \times \mathcal{P}(\mathcal{V}) \mid \xi \models \text{inv}(v)\}$, the set of edges E_u is defined as $E_\mathrm{u} = \{(v,\ g,\ l,\ (W,\ r),\ v') \in E \mid l \in L_\mathrm{u}\}$, and α_v is defined as the atomic interchange automaton α with initial location v and initial condition true: $\alpha_v = (X,\ X_\mathrm{i},\ \text{dtype},\ V,\ v,\ \text{true},\ \text{flow},\ \text{inv},\ \text{urgent},\ L,\ E)$.

7 Concluding remarks

The proposed interchange automaton format integrates formalisms rooted in computer science with those rooted in dynamics and control. Future work entails, among others, adding the notion of input/output variables and input/output actions, adding channels as communication mechanism between interchange automata in a parallel composition, adding additional operators, such as sequential composition, and defining the syntax of the concrete and transfer interchange formats as well as their mapping to the abstract interchange automaton format. The development of

translations and simulator implementations will be done by different partners in Work Package 3 of the HYCON NoE [21].

The atomic interchange automata as introduced syntactically in Definition 4.2 and for which the semantics has been introduced in Section 5 are very expressive. For any application of an action or variable hiding operator on an atomic interchange automaton, it is possible to obtain an equivalent atomic interchange automaton. Also, the parallel composition of any two atomic interchange automata for which the shared variables have compatible types can be replaced by an equivalent atomic interchange automaton. The only operator that cannot be eliminated in all relevant cases is the urgent action operator.

Bibliography

[1] R. Alur, T. A. Henzinger, and P. H. Ho. Automatic symbolic verification of embedded systems. *IEEE Transactions on Software Engineering*, 22(3):181–201, 1996.

[2] Rajeev Alur, Radu Grosub, Insup Lee, and Oleg Sokolsky. Compositional modeling and refinement for hierarchical hybrid systems. *Journal of Logic and Algebraic Programming*, 68(1-2):105–128, 2006.

[3] D. A. van Beek, K. L. Man, M. A. Reniers, J. E. Rooda, and R. R. H. Schiffelers. Syntax and consistent equation semantics of hybrid Chi. *Journal of Logic and Algebraic Programming*, 68(1-2):129–210, 2006.

[4] A. Benveniste, P. Caspi, S. A. Edwards, N. Halbwachs, P. Le Guernic, and R. de Simone. The synchronous languages 12 years later. *Proceeddings of the IEEE*, 91(1):64–83, January 2003.

[5] Henrik C. Bohnenkamp, Pedro R. D'Argenio, Holger Hermanns, and Joost-Pieter Katoen. Modest: A compositional modeling formalism for real-time and stochastic systems. Technical Report Technical Report TR-CTIT-04-46, University of Twente, Centre for Telematics and Information Technology, The Netherlands, 2004.

[6] Tommaso Bolognesi and Ferdinando Lucidi. Timed process algebras with urgent interactions and a unique powerful binary operator. In J. W. de Bakker, Cornelis Huizing, Willem P. de Roever, and Grzegorz Rozenberg, editors, *Real-Time: Theory in Practice, REX Workshop*, volume 600 of *Lecture Notes in Computer Science*, pages 124–148, Mook, The Netherlands, 1991. Springer-Verlag.

[7] Sébastien Bornot and Joseph Sifakis. An algebraic framework for urgency. *Information and Computation*, 163(1):172–202, 2000.

[8] K. E. Brenan, S. L. Campbell, and L. R. Petzold. *Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations*. Society for Industrial and Applied Mathematics, Philadelphia, 1996.

[9] Stefano Di Cairano, Alberto Bemporad, and Michal Kvasnica. An architecture for data interchange of switched linear systems. Technical Report D 3.3.1, HYCON NoE, 2006.

[10] Columbus IST project. http://www.columbus.gr, 2006.

[11] EcosimPro. http://www.ecosimpro.com, 2006.

[12] A. F. Filippov. *Differential Equations with Discontinuous Right Hand Sides*. Kluwer Academic Publishers, Dordrecht, 1988.

[13] Goran Frehse. PHAVer: Algorithmic verification of hybrid systems past HyTech. In Manfred Morari and Lothar Thiele, editors, *Hybrid Systems: Computation and Control, 8th International Workshop*, volume 3414 of *Lecture Notes in Computer Science*, pages 258–273. Springer-Verlag, 2005.

[14] Biniam Gebremichael and Frits Vaandrager. Specifying urgency in timed i/o automata. In *Proc. Third IEEE Conference on Sofware Engineering and Formal Methods*, pages 64–74, 2005.

[15] Nicolas Halbwachs. *Synchronous Programming of Reactive Systems*. Kluwer Academic, 1993.

[16] David Harel and Amnon Naamad. The statemate semantics of statecharts. *ACM Transactions on Software Engineering and Methodology*, 5(4):293–333, 1996.

[17] W. P. M. H. Heemels, B. de Schutter, and A. Bemporad. Equivalence of hybrid dynamical models. *Automatica*, 37(7):1085–1091, 2001.

[18] T. A. Henzinger. Masaccio: A formal model for embedded components. In Jan van Leeuwen, Osamu Watanabe, Masami Hagiya, Peter D. Mosses, and Takayasu Ito, editors, *Theoretical Computer Science, Exploring New Frontiers of Theoretical Informatics, International Conference IFIP TCS 2000, Sendai, Japan, August 17-19, 2000, Proceedings*, volume 1872 of *Lecture Notes in Computer Science*, pages 549–563. Springer-Verlag, 2000.

[19] 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.

[20] Thomas A. Henzinger, Pei-Hsin Ho, and Howard Wong-Toi. A user guide to HYTECH. In Ed Brinksma, Rance Cleaveland, Kim Guldstrand Larsen, Tiziana Margaria, and Bernhard Steffen, editors, *Tools and Algorithms for Construction and Analysis of Systems, First International Workshop, TACAS '95, Aarhus, Denmark, May 19-20, 1995, Proceedings*, volume 1019 of *Lecture Notes in Computer Science*, pages 41–71. Springer-Verlag, 1995.

[21] HYCON NoE. http://www.ist-hycon.org/, 2005.

[22] Y. Kesten and A. Pnueli. Timed and hybrid statecharts and their textual representation. In J. Vytopil, editor, *Formal Techniques in Real-Time and Fault-Tolerant Systems 2nd International Symposium*, volume 571 of *Lecture Notes in Computer Science*, pages 591–620, Nijmegen, The Netherlands, 1992. Springer-Verlag.

[23] Kim G. Larsen, Paul Pettersson, and Wang Yi. UPPAAL in a Nutshell. *International Journal on Software Tools for Technology Transfer*, 1(1–2):134–152, 1997.

[24] Nancy Lynch, Roberto Segala, and Frits Vaandrager. Hybrid I/O automata. *Information and Computation*, 185(1):105–157, 2003.

[25] K. L. Man and R. R. H. Schiffelers. *Formal Specification and Analysis of Hybrid Systems*. PhD thesis, Eindhoven University of Technology, 2006.

[26] Sven Erik Mattsson, Martin Otter, and Hilding Elmqvist. Modelica hybrid modeling and efficient simulation. In *38th IEEE Conference on Decision and Control*, pages 3502–3507, 1999.

[27] MoBIES team. HSIF semantics. Technical report, University of Pennsylvania, 2002. internal document.

[28] 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.

[29] X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. An approach to the description and analysis of hybrid systems. In R. L. Grossman, A. Nerode, A. P. Ravn, and H. Rischel, editors, *Hybrid Systems*, volume 736 of *Lecture Notes in Computer Science*, pages 149–178. Springer, 1993.

[30] Alessandro Pinto, Luca P. Carloni, Roberto Passerone, and Alberto L. Sangiovanni-Vincentelli. Interchange format for hybrid systems: Abstract semantics. In João P. Hespanha and Ashish Tiwari, editors, *Hybrid Systems: Computation and Control, 9th International Workshop*, volume 3927 of *Lecture Notes in Computer Science*, pages 491–506, Santa Barbara, 2006. Springer-Verlag.

[31] Gordon D. Plotkin. A structural approach to operational semantics. *Journal of Logic and Algebraic Programming*, 60-61:17–139, 2004.

[32] Manuel A. Pereira Remelhe and D. A. van Beek. Requirements for an interchange format for nonlinear hybrid systems. Technical Report D 3.6.1, HYCON NoE, 2006.

[33] A. J. van der Schaft and J. M. Schumacher. *An Introduction to Hybrid Dynamical Systems*, volume 251 of *Lecture Notes in Control and Information Sciences*. Springer-Verlag, 2000.

[34] The MathWorks, Inc. *Using Simulink, version 6*. http://www.mathworks.com, 2005.

[35] Michael Tiller. *Introduction to Physical Modeling with Modelica*, volume 615 of *The International Series in Engineering and Computer Science*. Springer-Verlag, 2001.

[36] Uppsala University and Aalborg University. *UPPAAL version 4.0 online help*, 2006.

[37] Michael von der Beeck. A comparison of statecharts variants. In Hans Langmaack, Willem P. de Roever, and Jan Vytopil, editors, *Formal Techniques in Real-Time and Fault-Tolerant Systems, Third International Symposium*, volume 863 of *Lecture Notes in Computer Science*, pages 128–148, Lübeck, Germany, 1994. Springer-Verlag.