Cottbus Timed Automata: Formal Definition and Compositional Semantics

Dirk Beyer and Heinrich Rust

Software Systems Engineering Research Group
Technical University Cottbus, Germany
{db|rust}@informatik.tu-cottbus.de

Abstract. We present a formalism for modular modeling of hybrid systems, the Cottbus Timed Automata. For the theoretical basis, we build on work about timed and hybrid automata. We use concepts from concurrency theory to model communication of separately defined modules, and we extend these concepts to be able to express explicitly read- and write-access to signals and variables.

1 Introduction

The programming of embedded systems which have to fulfill hard real-time requirements is becoming an increasingly important task in different application areas, e.g. in medicine, in transport technology or in production automation. The application of formal methods, i.e. of modeling formalisms and analysis methods having a sound mathematical basis, is expected to lead to the development of systems with less defects via a better understanding of critical system properties (cf. (Rust, 1994)).

In (Beyer and Rust, 1998) a modeling notation is presented which allows to model hybrid systems in a modular way. It builds on the theoretical basis used in tools like Uppaal (Bengtsson *et al.*, 1996), Kronos (Daws *et al.*, 1996) and HyTech (Henzinger *et al.*, 1995). In these formalisms and tools, finite automata are used to model the control component of an automaton, and analogue variables which may vary continuously with time are used to model the non-discrete system components of a hybrid system. Partial automata of a larger system communicate via CSP-like synchronization labels (Hoare, 1985). Algorithms for these kinds of models have been presented in (Alur *et al.*, 1993) and (Henzinger *et al.*, 1994).

On the basis of the refined notation presented in (Beyer and Rust, 1998), we introduce the following concepts:

- Compositional semantics: By defining the semantics of a hybrid automaton as labeled update system in our formalism we preserve the information we need to define the semantics of a CTA module on the basis of the semantics of its parts.
- Hierarchy: Subsystem descriptions can be grouped. Interfaces and local components are separated.
- Explicit handling of different types of communication signals: We allow to express explicitly that an event is an input signal for an automaton, an output signal, or a multiply restricted signal.
- We allow to express explicitly that an analogue variable is accessed by an automaton as output, as input, or that it is multiply restricted.
- Automatical completion of automata for input signals. Input signals are events which an automaton must always admit. If there are configurations of an automaton in which the reaction to an input signal is not defined, it is understood that the automaton enters an error location.
- Recurring subsystem components have not to be multiply defined. They are instantiated from a common module type.

The differentiation between different roles of signals in an automaton has been used in the definition of IO-automata (Lynch and Tuttle, 1987) and extended to hybrid systems in (Lynch *et al.*, 1996). In (Alur and Henzinger, 1997), another approach is presented to describe modular hybrid systems. It builds on reactive modules (Alur and Henzinger, 1996) and extends them with continuously changing variables.

2 Formal Definition

Our semantics for the basic underlying model is similar to that of hybrid automata (Henzinger, 1996). We extend it by formalizing our concepts of different types of input, output, multiply restricted and locally defined signals and variables, and by formalizing what it means to instantiate a module in a context module.

2.1 Hybrid Automata

At first, we define value assignments, which are used to define invariants and derivations of locations, as well as guards and value changes of transitions of a hybrid automaton. Then we introduce some helpful notions and abbreviations, and the definition of hybrid automata follows.

Notation. Let $V = \{v_1, \dots, v_n\}$ be a finite set of variables. A **value assignment** $a : V \to \mathbb{R}$ is a total function from V into $\mathbb{R}$, where $\mathbb{R}$ is the set of real numbers. We write $\mathbb{R}_+$ for the set of all non-negative real numbers. $\mathbb{N}$ denotes the set of natural numbers, 0 inclusive. $A(V)$ denotes the set of all value assignments of V. For a finite set S, a pair $c \in C = S \times A(V)$ is called **configuration**. It consists of an element of S and a value assignment. Sometimes, a set of configurations is called a **region**. $(q_i)_{i \in \mathbb{N}}$ abbreviates an infinite sequence of elements $(q_0, q_1, q_2, \dots)$.

For an S-tuple of length n, $\pi_i : S_1 \times \dots \times S_n \to S_i$ with $i \in \{1, \dots, n\}$ is a **projection** operator: it selects the i'th element: $\pi_i((s_1, \dots, s_n)) = s_i$. For a partial function $f : D \to\!\!\!\!\!\!\circ\ \ R$, we write $\text{dom}(f)$ for its **domain**: $\text{dom}(f) =_{\text{def}} \{x \in D | \exists y \in R : y = f(x)\}$, and $\text{range}(f)$ for its **range**: $\text{range}(f) =_{\text{def}} \{y \in R | \exists x \in D : y = f(x)\}$. For a partial function $f : D \to\!\!\!\!\!\!\circ\ \ R$ and a set $D' \subseteq D$, the **restriction** of f to D' is the function $f|_{D'} : D' \to\!\!\!\!\!\!\circ\ \ R$ with $\forall x \in D' \cap \text{dom}(f) : f|_{D'}(x) = f(x)$. For a set F of functions, $F|_{D'}$ is the result of applying $\cdot|_{D'}$ component-wise.

For a total function $f : D \to R$ and a partial function $g : D \to\!\!\!\!\!\!\circ\ \ R$ we define the **override** operator $\lhd : ((D \to R) \times (D \to\!\!\!\!\!\!\circ\ \ R)) \to (D \to R)$ as follows:

$$(f \lhd g)(x) =_{\text{def}} \begin{cases} g(x) \text{ if } x \in dom(g) \\ f(x) \text{ otherwise} \end{cases}$$

We define the **extension** of a value assignment or a configuration by not restricting the values for the added variables, and some further notation. Let $C \subseteq S \times A(V)$ be a set of configurations over the locations S and the variable assignments for V, let $B \subseteq A(V)$ be a set of variable assignments for V, and let V' be a superset of V. Then, the extensions of B and C to V', written $\text{extend}(B, V')$ and $\text{extend}(C, V')$, are defined as follows:

$$\begin{aligned} &\text{extend}(B, V') =_{\text{def}} \{a' \in A(V') | \exists a \in B : a = (a'|_V)\} \\ &\text{extend}(C, V') =_{\text{def}} \{(s', a') \in S \times A(V') | \exists (s, a) \in C : s = s' \wedge a = (a'|_V)\} \end{aligned}$$

We define the **extended union** operator $\dot\cup : 2^{A(V)} \times 2^{A(V')} \to 2^{A(V \cup V')}$ for two sets of value assignments $B \subseteq A(V)$ and $B' \subseteq A(V')$ as $B \dot\cup B' =_{\text{def}} \text{extend}(B, V \cup V') \cup \text{extend}(B', V \cup V')$. For the **extended intersection** $\dot\cap : 2^{A(V)} \times 2^{A(V')} \to 2^{A(V \cup V')}$ we use a similar definition: $B \dot\cap B' =_{\text{def}} \text{extend}(B, V \cup V') \cap \text{extend}(B', V \cup V')$. Analogously we define union and intersection for sets of configurations.

The basic concept in our formalization is the hybrid automaton.

Definition 1. (Hybrid automata) *A* **hybrid automaton** *consists of the following components:*

- S*: A finite set of control locations.*
- G*: A finite set of signals.*
- V*: A finite set of analogue variables. Variables are used in* **value assignments**.
- $I \subseteq (S \times A(V))$*: An initial condition, described as a set of configurations.*
- T*: A finite set of transitions.*
- $\mathit{inv} : S \to 2^{A(V)}$*: A function associating an invariant to each location. The invariant is a set of value assignments.*
- $\mathit{deriv} : S \to 2^{A(V)}$*: A function associating a set of value assignments to each location. We interpret the values given by these value assignments as admissible time derivatives for the variables while the automaton is in the argument location.*
- $\mathit{trans} : T \to S \times S$*: A function associating a source location and a target location to each transition.*
- $\mathit{guard} : T \to 2^{A(V)}$*: A function associating a guard with each transition.*
- $\mathit{sync} : T \to G \cup \{*\}$*: A function associating a signal or no signal to each transition.* $*$ *is not a signal; it is the value of* $\mathit{sync}(t)$ *for transitions without a signal.*
- $\mathit{update} : T \to (A(V) \to 2^{V \,-\!\!\ominus\!\!\!\rightarrow\, \mathbb{R}})$*: A function associating with each transition a function which yields for a value assignment a set of updates. The aim of this function is to define value changes for the variables. An update is a value assignment for a subset of the variables of the automaton. For a particular update* $u \in \mathit{update}(t)(a)$ *of transition* t *and value assignment* a*, the variables from* $V \setminus \mathit{dom}(u)$ *are not changed.*

  For each $t \in T$ *and* $a \in \mathit{guard}(t)$*, the set* $\mathit{update}(t)(a)$ *must be nonempty. This condition ensures that the 'update' component can not inhibit a discrete transition to be taken.*

A further restrictions is: For each $s \in S$*, there is an element* t_s *of* T *with the following properties:*

- $\mathit{trans}(t_s) = (s, s)$
- $\mathit{guard}(t_s) = A(V)$
- $\mathit{sync}(t_s) = *$
- $\forall a \in A(V) : \forall u \in \mathit{update}(t_s)(a) : \mathrm{dom}(u) = \emptyset$

These transitions are no-op transitions. The subset of T *consisting of all no-op transitions is referred to by 'noop'. The 'update' function of no-op transitions does not restrict the environment in any way, i.e. all variables stay unchanged.*

Note. 'inv', 'deriv' and 'update' are typically defined via predicates over the variables of the automaton. For 'update' predicates, non-primed variables represent values of the function argument (before the transition) and primed variables represent values of the function result (after the transition). In the domain of the 'update' functions only those variables occur which are restricted by the corresponding predicate.

If there is a variable which does not occur primed in at least one inequation of the 'update' predicate, then this does not mean that the whole range of $\mathbb{R}$ is possible. For a variable which does not occur primed in the 'update' predicate the interpretation is that this transition does not change the value of the variable, but transitions of environmental automata are allowed to restrict the variable. But if no automaton restricts the variable v in its transition in the same point in time, then it does not change its value. To express that the whole range of $\mathbb{R}$ is possible for v after a transition even without a context, one might use the 'update' predicate $\{v' > 0 \text{ OR } v' \leq 0\}$.

In the sequel of the paper we use the dot notation $A.x$ to address the component x of A.

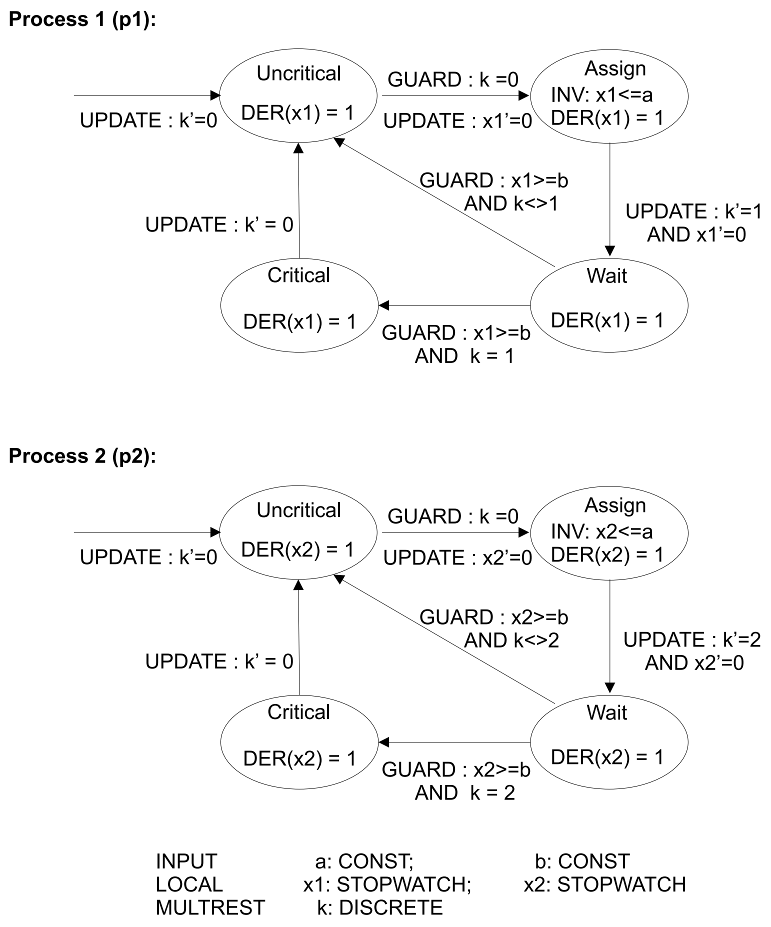

Fig. 1. Fischer's mutual exclusion protocol

2.2 Illustration

To show the intention of the hybrid automata we display the automata for Fischer's timing-based mutual exclusion protocol in Figure 1 (cf. (Lamport, 1987)). In our Fischer automaton a process is modeled by four locations. The initial location is the location Uncritical, which models the uncritical section, and the initial value of the shared variable k is 0. From this location only one transition is possible: If the shared variable signifies that no process is in the critical section then the process can try to enter the critical section. It enters the location modeling the Assign statement. The clock x_i (with time derivation 1 in all locations) measures the time staying in this location, and the invariant forces to leave the location after at most time a, which models the maximal time needed by the assign statement of the process. Then the transition to the Wait location sets the variable k to the number of the process. In this location we have to wait at least time b to give other processes a chance to set k to its process number. After time b the process can decide to enter the critical section if k = i. Otherwise it goes back to the uncritical section. Leaving the critical section the automaton sets k to value 0 to signify that the resource is free again.

To present our notation we show the textual version of the system in Figure 2. Implicit invariants and guards are understood to be true. The instantiation concept used in the module is explained in the sequel of the paper.

The parallel composition of several hybrid automata and the semantics of a hybrid automaton will be defined formally in the next two sections.

```

MODULE Process {
  INPUT
    // Constants for time bounds.
    a:          CONST;  // a is the maximal time the modeled assignment k:=1 needs.
    b:          CONST;  // b is the minimal time the process waits for assignments
                        //   initiated from other processes.
    processNo:  CONST;  // Parameter for significant value of k.
  MULTREST
    k:  DISCRETE;       // k is the shared variable for announcement.
  LOCAL
    x:  CLOCK;          // A clock to measure the time in a state.

  INITIALIZATION { STATE(Fischer) = uncritical AND k = 0; }

  AUTOMATON Fischer {
    STATE uncritical { DERIV { DER(x) = 1;}
                       TRANS assign     { GUARD  { k  = 0; }
                                          UPDATE { x'= 0; } } }
    STATE assign     { INV { x <= a; }
                       DERIV { DER(x) = 1; }
                       TRANS wait       { UPDATE { x'= 0 AND
                                                   k'= processNo; } } }
    STATE wait       { DERIV { DER(x) = 1; }
                       TRANS uncritical { GUARD { x >= b AND
                                                  k <>processNo; } }
                       TRANS critical   { GUARD { x >= b AND
                                                  k = processNo; } } }
    STATE critical   { DERIV { DER(x) = 1; }
                       TRANS uncritical { UPDATE { k'= 0; } } }
  }
}

MODULE System {
  LOCAL
    a = 3: CONST;      b = 3: CONST;
    pNo1 = 1: CONST;   pNo2 = 2: CONST;
  MULTREST
    k:     DISCRETE;

  INST Process1 FROM Process WITH {
    a         AS a;
    b         AS b;
    processNo AS pNo1;
    k         AS k;
  }
  INST Process2 FROM Process WITH {
    a         AS a;
    b         AS b;
    processNo AS pNo2;
    k         AS k;
  }
}

```

Fig. 2. Fischer's Protocol: An example for a CTA model

2.3 Parallel Composition of Hybrid Automata

In this section, we define what it means for two hybrid automata to be composed parallelly. Modular specification makes it necessary to combine several hybrid automata.

Definition 2. (Parallel composition of hybrid automata) *Let $\mathcal{H}$ and $\mathcal{H}'$ be two hybrid automata. Their parallel composition $\mathcal{P} = \mathcal{H}||\mathcal{H}'$ is defined in the following way (cf. (Henzinger, 1996) for the general principle):*

- $\mathcal{P}.S =_{def} \quad \mathcal{H}.S \times \mathcal{H}'.S$
- $\mathcal{P}.G =_{def} \quad \mathcal{H}.G \cup \mathcal{H}'.G$
- $\mathcal{P}.V =_{def} \quad \mathcal{H}.V \cup \mathcal{H}'.V$
- $\mathcal{P}.I =_{def} \quad \mathcal{H}.I \quad ⩀ \quad \mathcal{H}'.I$
- $\mathcal{P}.T =_{def} \quad \left\{ (t,t') \in \mathcal{H}.T \times \mathcal{H}'.T \;\middle|\; \begin{array}{l} \mathcal{H}.sync(t) = \mathcal{H}'.sync(t') \neq * \\ \vee \quad (\mathcal{H}.sync(t) \notin \mathcal{H}'.G \wedge t' \in \mathcal{H}'.noop) \\ \vee \quad (\mathcal{H}'.sync(t') \notin \mathcal{H}.G \wedge t \in \mathcal{H}.noop) \end{array} \right\}$
- $\mathcal{P}.inv((s,s')) =_{def} \quad \mathcal{H}.inv(s) \quad ⩀ \quad \mathcal{H}'.inv(s')$
- $\mathcal{P}.deriv((s,s')) =_{def} \quad \mathcal{H}.deriv(s) \quad ⩀ \quad \mathcal{H}'.deriv(s')$
- $\mathcal{P}.trans((t,t')) =_{def}$
  $(\quad (\pi_1(\mathcal{H}.trans(t)), \pi_1(\mathcal{H}'.trans(t'))), \quad (\pi_2(\mathcal{H}.trans(t)), \pi_2(\mathcal{H}'.trans(t'))) \quad)$

– $\mathcal{P}.sync((t,t')) =_{def} \begin{cases} \mathcal{H}'.sync(t') \text{ if } \mathcal{H}.sync(t) = * \\ \mathcal{H}.sync(t) \text{ otherwise} \end{cases}$

– $\mathcal{P}.update((t,t'))(a) =_{def}$

$$\left\{ u : \mathcal{P}.V \rightharpoonup \mathbb{R} \;\middle|\; \begin{array}{rl} & \exists u_1 \in \mathcal{H}.update(t)(a|_{\mathcal{H}.V}) : \quad u|_{\mathrm{dom}(u_1)} = u_1 \\ \wedge & \exists u_2 \in \mathcal{H}'.update(t')(a|_{\mathcal{H}'.V}) : u|_{\mathrm{dom}(u_2)} = u_2 \end{array} \right\}$$

– $\mathcal{P}.guard((t,t')) =_{def}$

$$\mathcal{H}.guard(t) \quad \Cap \quad \mathcal{H}'.guard(t') \quad \Cap \quad \{a \in A(\mathcal{P}.V) \quad | \quad \mathcal{P}.update((t,t'))(a) \neq \{\}\}$$

Note. The set of transitions of the parallel composition consists of pairs of transitions which have to be executed together. The no-op transitions defined to be in $\mathcal{H}.T$ ensure that independent transitions in the two automata can be executed independently in the parallel composition. For non-synchronizing pairs of transitions, at least one of the paired transitions must be a no-op transition.

We include the information in the guard that the resulting set of 'update' functions for a value assignment is empty or nonempty. In this way, the restriction for 'update' sets in hybrid automata, i.e. that they have to be nonempty for value assignments in the guard, is trivially fulfilled for transitions with contradicting 'update' functions.

The parallel composition allows all transformations of value assignments which are allowed by $\mathcal{H}$.update and $\mathcal{H}$'.update, i.e. they must not be contradictive. If we do not restrict a variable in the 'update' set of a transition, this means that the value is not changed. Note that the set of 'update' functions which does not change any variables is the singleton set of functions which contains only a function nowhere defined (i.e. all update functions with empty domain). The contradictive set of update functions is the empty set (i.e. there is no update function, the domain does not matter).

The parallel composition of two hybrid automata is again a hybrid automaton:

Proposition 1. *Let $\mathcal{H}$ and $\mathcal{H}'$ be two hybrid automata. Then $\mathcal{H}||\mathcal{H}'$ is also a hybrid automaton.*

Notation. (Parallel composition of several automata) Let $\mathcal{A}$ be a nonempty finite set of hybrid automata. Then $\prod_{a\in\mathcal{A}} a$ denotes a parallel composition of all elements of $\mathcal{A}$ in some order.

Note. Different orders of the automata from $\mathcal{A}$ lead to different automata, but with respect to the communication behavior they are isomorphic.

2.4 Compositional Semantics of Hybrid Automata

This section defines labeled update systems and how to construct a labeled update system for a given hybrid automaton. We define also the trace semantics and we provide labeled transition systems as intermediate abstraction level.

The semantics of a dynamical system is often defined as a transition system. For systems which allow synchronous composition, a transition system does not provide enough structure: a labeled transition system semantics is not compositional for synchronous composition. CTA have a synchronous composition operator. Because of that, we define an update semantics, which keeps enough structure to allow the definition of synchronous composition.

Definition 3. *A* **labeled update system** *is a tuple $(V, \mathcal{U}, Q_0, \Sigma, *, U)$, where*

– *V is a set of variables.*
– *$\mathcal{U}$ is a universe of values.*
  Q is the state space of the labeled update system. It is defined by V and $\mathcal{U}$ together: $Q =_{def} V \to \mathcal{U}$. Each state is an assignment which maps each variable to a value from the universe.
– *$Q_0 \subseteq Q$ is the set of initial states.*

- *Σ is the set of labels.*
- *$* \in \Sigma$ is a special label for an update without synchronization.*
- *$U \subseteq (V \to \mathcal{U}) \times \Sigma \times (V \nrightarrow \mathcal{U})$ are the updates of the labeled update system. $(q, \sigma, u) \in U$ means that in state q on label σ, the update described by u can take place, which means that each variable v in the domain of u gets the value $u(v)$, and other variables stay unchanged.*

Notation. For a real δ and two value assignments a and $a' \in A(V)$, let $\delta * a$ denote the function $\lambda(v : V) : \delta * a(v)$, and let $a + a'$ denote the function $\lambda(v : V) : a(v) + a'(v)$. The passage of some time $\delta \in \mathbb{R}_+$ together with a fixed time derivative $d \in A(V)$ for the time-dependent value changes of variables, leads from a configuration (s, a) to the resulting configuration $(s, a + \delta * d)$.

The labeled update system corresponding to a hybrid automaton is defined in the following way:

Definition 4. *Let $\mathcal{H}$ be a hybrid automaton. The* **labeled update system** *$us(\mathcal{H}) = (V, \mathcal{U}, Q_0, \Sigma, *, U)$ corresponding to $\mathcal{H}$ is defined in the following way:*

- *$V =_{def} \mathcal{H}.V \cup \{s_{\mathcal{H}}\}$, $s_{\mathcal{H}}$ is a new element which is characteristic for $\mathcal{H}$: it is a variable representing the location of automaton $\mathcal{H}$.*
- *$\mathcal{U} =_{def} \mathbb{R} \cup \mathcal{H}.S$*
- *$Q_0 =_{def} \{q \in V \to \mathcal{U} \mid (q(s_{\mathcal{H}}), q|_{\mathcal{H}.V}) \in \mathcal{H}.I\}$*
- *$\Sigma =_{def} \mathcal{H}.G \cup \mathbb{R}_+ \cup \{*\}$*
- *$U =_{def}$ time$(\mathcal{H}) \cup$ discrete$(\mathcal{H})$ with:*

$$\mathit{time}(\mathcal{H}) =_{def} \left\{ (q,\sigma,u) \;\middle|\; \begin{array}{l} \exists\, d \in \mathcal{H}.\mathit{deriv}(q(s_{\mathcal{H}})) : \exists \delta \in (\mathbb{R}_+ \setminus 0) : \\ \quad u(s_{\mathcal{H}}) = q(s_{\mathcal{H}}) \\ \wedge\, u|_{\mathcal{H}.V} = q|_{\mathcal{H}.V} + \delta * d \\ \wedge\, \forall(\delta' : 0 \le \delta' \le \delta) : q|_{\mathcal{H}.V} + \delta' * d \in \mathcal{H}.\mathit{inv}(q(s_{\mathcal{H}})) \\ \wedge\, \sigma = \delta \end{array} \right\}$$

$$\mathit{discrete}(\mathcal{H}) =_{def} \left\{ (q,\sigma,u) \;\middle|\; \begin{array}{l} \exists\, t \in \mathcal{H}.T : \\ \quad \mathcal{H}.\mathit{trans}(t) = (q(s_{\mathcal{H}}), u(s_{\mathcal{H}})) \\ \wedge\, q|_{\mathcal{H}.V} \in \mathcal{H}.\mathit{guard}(t) \\ \wedge\, u|_{\mathcal{H}.V} \in \mathcal{H}.\mathit{update}(t)(a|_{\mathcal{H}.V}) \\ \wedge\, \sigma = \mathcal{H}.\mathit{sync}(t) \end{array} \right\}$$

Note. The state space of the labeled update system consists of the configurations of the hybrid automaton. The set of starting states of the labeled update system is defined via the initial condition of the hybrid automaton. A hybrid automaton can perform time updates and discrete updates, thus the updates of the labeled update system are all time updates and all discrete updates.

Illustration. We illustrate these definitions with our example in Figure 1: If the automaton p1 has entered location Assign then the variable x1 has the value 0 and the first time derivative of x1 is 1. In this situation the following **time updates** of the automaton are possible: For each time u in the interval $(0, \mathsf{a}]$ (a is the upper bound for x1 in the invariant) the automaton may take the time update to the configuration (Assign, q) with $q(\mathsf{x1}) = \mathsf{x1} + u * 1$ (q is a value assignment for the automaton, $q : \mathcal{H}.V \to \mathbb{R}$). The other possibility is to take the **discrete update** choosing the transition to location Wait leading to the configuration (Wait, q) where q is not changed by the transition.

Note. For discrete updates the invariant is irrelevant. An invariant which is identically false can be used to construct urgent locations, i. e. locations in which time cannot pass. The location component of the configuration may not change in a time update.

Update systems contain enough structure for the definition of synchronous composition:

Definition 5. *Let $\mathcal{S}, \mathcal{S}'$ be two labeled update systems. Their synchronous composition $\mathcal{S}||\mathcal{S}'$ is defined as $(V, \mathcal{U}, Q_0, \Sigma, *, U)$, where*

- $V =_{def} \mathcal{S}.V \cup \mathcal{S}'.V$
- $\mathcal{U} =_{def} \mathcal{S}.\mathcal{U} \times \mathcal{S}'.\mathcal{U}$
- $Q_0 =_{def} \{q \in (V \to \mathcal{U}) \quad | \quad q|_{\mathcal{S}.V} \in \mathcal{S}.Q_0 \wedge q|_{\mathcal{S}'.V} \in \mathcal{S}'.Q_0\}$
- $\Sigma =_{def} \mathcal{S}.\Sigma \cup \mathcal{S}'.\Sigma$
- $U =_{def} \left\{ (q, \sigma, u) \;\middle|\; \begin{array}{l} \exists(q_1, \sigma_1, u_1) \in \mathcal{S}.U : \quad \exists(q_2, \sigma_2, u_2) \in \mathcal{S}'.U : \\ \quad q|_{\mathcal{S}.V} = q_1 \\ \wedge \;\; q|_{\mathcal{S}'.V} = q_2 \\ \wedge \;\; u|_{\mathrm{dom}(u_1)} = u_1 \\ \wedge \;\; u|_{\mathrm{dom}(u_2)} = u_2 \\ \wedge \;\; \left(\begin{array}{l} \quad (\sigma = \sigma_1 = \sigma_2 \neq *) \\ \vee\, (\sigma \notin \mathcal{S}'.\Sigma \wedge \sigma = \sigma_1 \wedge \sigma_2 = * \wedge \mathrm{dom}(u_2) = \emptyset) \\ \vee\, (\sigma \notin \mathcal{S}.\Sigma \wedge \sigma = \sigma_2 \wedge \sigma_1 = * \wedge \mathrm{dom}(u_1) = \emptyset) \end{array} \right) \end{array} \right\}$

Note that in the labeled transition system, for unchanged values in a transition one does not know whether the variable was unchanged or has been updated explicitly to the value it had before, while in an update system this information is available. This is the reason why an update system semantics is compositional for synchronous composition, but not a labeled transition system semantics. We use the same symbol $||$ as composition operator for hybrid automata as well as for labeled update systems.

The parallel composition of two labeled update systems is again a labeled update system:

Proposition 2. *Let $\mathcal{S}$ and $\mathcal{S}'$ be two labeled update systems. Then $\mathcal{S}||\mathcal{S}'$ is also a labeled update system.*

Proposition 3. *The labeled update system semantics is compositional for hybrid automata, i.e. for two hybrid automata $\mathcal{H}$ and $\mathcal{H}'$, $us(\mathcal{H}||\mathcal{H}') = us(\mathcal{H})||us(\mathcal{H}')$.*

Proof. The proof follows from the definitions of the construction operators $||$. □

To define also the (traditional) semantics as labeled transition system for hybrid automata we have to introduce this notion.

Definition 6. *A* **labeled transition system** *is a tuple $(Q, Q_0, \Sigma, *, T)$, where:*

- *Q is a (possibly infinite) set of states,*
- *$Q_0 \subseteq Q$ is a set of initial states,*
- *A set Σ of labels.*
- *A set $T \subseteq Q \times \Sigma \times Q$ of transitions.*

We can construct the labeled transition system $\mathrm{ts}(\mathrm{us}(\mathcal{H}))$ corresponding to a hybrid automaton $\mathcal{H}$ from its labeled update system $\mathrm{us}(\mathcal{H})$:

Definition 7. *Let $\mathcal{S}$ be the labeled update system. The* **labeled transition system** *$ts(\mathcal{S}) = (Q, Q_0, \Sigma, *, T)$ is defined in the following way:*

- $Q =_{def} \mathcal{S}.Q$
- $Q_0 =_{def} \mathcal{S}.Q_0$
- $\Sigma =_{def} \mathcal{S}.\Sigma$
- $T =_{def} \{(q, \sigma, q \lhd u) \in Q \times \Sigma \times Q \quad | \quad (q, \sigma, u) \in \mathcal{S}.U\}$

Note. The state space of the labeled transition system is the state space of the labeled update system. The set of initial states of the transition system is defined by the initial states of the labeled update system. The transitions of the labeled transition system are computed by applying the update function.

Notation. A **run** for a labeled transition system $\mathcal{T} = (Q, Q_0, \Sigma, *, T)$ is an infinite sequence $(q_i, \sigma_i)_{i \in \mathbb{N}}$ with $q_0 \in Q_0$ and $\forall i \in \mathbb{N} : q_i \in Q, \sigma_i \in \Sigma \setminus \{*\}, (q_i, \sigma_i, q_{i+1}) \in T^+$. T^+ is the closure regarding non-synchronizing transitions; formally we define T^+ inductively, for $\sigma \neq *$:

– $(q,\sigma,q') \in T \Rightarrow (q,\sigma,q') \in T^+$
– $(q,\sigma,q'') \in T^+ \wedge (q'',*,q') \in T \Rightarrow (q,\sigma,q') \in T^+$
– $(q,*,q'') \in T \wedge (q'',\sigma,q') \in T^+ \Rightarrow (q,\sigma,q') \in T^+$

For a given labeled transition system $\mathcal{T}$, the infinite sequence $(\sigma_i)_{i\in\mathbb{N}}$ is a **trace** iff there exists a corresponding run $(q_i,\sigma_i)_{i\in\mathbb{N}}$ of $\mathcal{T}$.

Note. The transition relation T contains so-called silent transitions, i.e. transitions without synchronization. These transitions are not considered by the trace semantics, because they do not influence the environment.

Definition 8. *The* **trace semantics** $\mathcal{L}(\mathcal{T})$ *is the set of all traces of the labeled transition system* $\mathcal{T}$*. For two trace semantics* $\mathcal{L}(\mathcal{T})$ *and* $\mathcal{L}(\mathcal{T}')$*, composition is defined by intersection:* $\mathcal{L}(\mathcal{T})||\mathcal{L}(\mathcal{T}') =_{def} \mathcal{L}(\mathcal{T}) \cap \mathcal{L}(\mathcal{T}')$.

We can construct the trace semantics $\mathcal{L}(\text{ts}(\mathcal{S}))$ for a labeled update system $\mathcal{S}$ from its labeled transition system $\text{ts}(\mathcal{S})$. We write the abbreviation $\mathcal{L}(\mathcal{S})$ for $\mathcal{L}(\text{ts}(\mathcal{S}))$.

Proposition 4. *Let* $\mathcal{S}^1 = (V^1,\mathcal{U}^1,Q_0^1,\Sigma^1,*,U^1)$ *and* $\mathcal{S}^2 = (V^2,\mathcal{U}^2,Q_0^2,\Sigma^2,*,U^2)$ *be two labeled update systems with* $V^1 \cap V^2 = \emptyset$ *and* $\Sigma^1 = \Sigma^2$*. Then the trace semantics is compositional for the operator* $||$ *:* $\mathcal{L}(\mathcal{S}^1)||\mathcal{L}(\mathcal{S}^2) = \mathcal{L}(\mathcal{S}^1||\mathcal{S}^2)$.

Proof. $\subseteq$: We start to show $\mathcal{L}(\mathcal{S}^1)||\mathcal{L}(\mathcal{S}^2) \subseteq \mathcal{L}(\mathcal{S}^1||\mathcal{S}^2)$. From $\alpha = (\sigma_i)_{i\in\mathbb{N}} \in \mathcal{L}(\mathcal{S}^1)||\mathcal{L}(\mathcal{S}^2)$ it follows by the definition of the composition of two trace sets as intersection $\alpha \in \mathcal{L}(\mathcal{S}^1)$ and $\alpha \in \mathcal{L}(\mathcal{S}^2)$. Thus, we find two runs $(q_i^1,\sigma_i)_{i\in\mathbb{N}}$ of $\mathcal{S}^1$ and $(q_i^2,\sigma_i)_{i\in\mathbb{N}}$ of $\mathcal{S}^2$. Since both systems communicate with each other only via (the same set of) labels but not via shared variables ($V^1 \cap V^2 = \emptyset$), we can construct a run $(q_i,\sigma_i)_{i\in\mathbb{N}}$ where $q_i|_{V^1} = q_i^1$ and $q_i|_{V^2} = q_i^2$. Because the composition of two updates of a labeled update system is built exactly in this way, we can conclude that $(q_i,\sigma_i)_{i\in\mathbb{N}}$ is a run of $\mathcal{S}^1||\mathcal{S}^2$ and thus, $\alpha \in \mathcal{L}(\mathcal{S}^1||\mathcal{S}^2)$.

$\supseteq$: It remains to be shown that $\mathcal{L}(\mathcal{S}^1)||\mathcal{L}(\mathcal{S}^2) \supseteq \mathcal{L}(\mathcal{S}^1||\mathcal{S}^2)$. For each trace $\alpha \in \mathcal{L}(\mathcal{S}^1||\mathcal{S}^2)$ exists a run $(q_i,\sigma_i)_{i\in\mathbb{N}}$ of $\mathcal{S}^1||\mathcal{S}^2$. Because $V^1 \cap V^2 = \emptyset$, we can split $q_i : V^1 \cup V^2 \to \mathbb{R}$ into $q_i^1 : V^1 \to \mathbb{R}$ and $q_i^2 : V^2 \to \mathbb{R}$ with $q_i|_{V^1} = q_i^1$ and $q_i|_{V^2} = q_i^2$. Per induction this is sound for all q_i. Therefore we can construct two runs $(q_i^1,\sigma_i)_{i\in\mathbb{N}}$ of $\mathcal{S}^1$ and $(q_i^2,\sigma_i)_{i\in\mathbb{N}}$ of $\mathcal{S}^2$. Thus we get $\alpha \in \mathcal{L}(\mathcal{S}^1)$ and $\alpha \in \mathcal{L}(\mathcal{S}^2)$. □

Note. Without the condition $V^1 \cap V^2 = \emptyset$ the proposition is not true for sets of traces. As counterexample we consider a trace $\alpha = (\sigma_i)_{i\in\mathbb{N}} \in \mathcal{L}(\mathcal{S}^1)||\mathcal{L}(\mathcal{S}^2)$ and there exist only runs $(q_i^1,\sigma_i)_{i\in\mathbb{N}}$ of $\mathcal{S}^1$ and $(q_i^2,\sigma_i)_{i\in\mathbb{N}}$ of $\mathcal{S}^2$ with $q_k^1(v) \neq q_k^2(v)$ for some k and $v \in V^1 \cap V^2$. Then there exists no run $(q_i,\sigma_i)_{i\in\mathbb{N}}$ of $\mathcal{S}^1||\mathcal{S}^2$, and $\alpha \notin \mathcal{L}(\mathcal{S}^1||\mathcal{S}^2)$.

$\Sigma^1 = \Sigma^2$ is a necessary condition. Let $\Sigma^1 \setminus \Sigma^2 \neq \emptyset$. Then the reaction of system $\mathcal{S}^1$ on label $\sigma \in \Sigma^1 \setminus \Sigma^2$ is allowed in the composed system $\mathcal{S}^1||\mathcal{S}^2$, but it cannot be an element of the intersection of the traces $\mathcal{L}(\mathcal{S}^1)$ and $\mathcal{L}(\mathcal{S}^2)$ because $\mathcal{S}^2$ has no trace in which σ occurs.

2.5 Hybrid Modules

Hybrid modules are hybrid automata with partitions of the variables and signals into input, output, multiply restricted and local sets. Thus, hybrid modules encapsulate those of our new concepts which concern the interface specification of a CTA module.

Definition 9. *A* **hybrid module** *consists of the following components:*

– $\mathcal{H}$*: A hybrid automaton.*
– $G = \mathcal{H}.G$*: The signals of the hybrid module are those of the hybrid automaton.*
– $GI \subseteq \mathcal{H}.G$*: The set of input signals.*
– $GO \subseteq \mathcal{H}.G$*: The set of output signals.*

– *$GM \subseteq \mathcal{H}.G$: The set of multiply restricted signals.*
– *$GL \subseteq \mathcal{H}.G$: The set of locally defined signals.*
– *$V = \mathcal{H}.V$: The variables of the hybrid module are those of the hybrid automaton.*
– *$VI \subseteq \mathcal{H}.V$: The set of input variables.*
– *$VO \subseteq \mathcal{H}.V$: The set of output variables.*
– *$VM \subseteq \mathcal{H}.V$: The set of multiply restricted variables.*
– *$VL \subseteq \mathcal{H}.V$: The set of locally defined variables.*

These components have to fulfill the following axioms:

– *GI,GO,GM and GL are a partition of $\mathcal{H}.G$. This means that they are pairwise disjoint, and their union is $\mathcal{H}.G$.*
– *For each $s \in \mathcal{H}.S$ and each $g_i \in GI$, the following holds: Let $T' \subseteq \mathcal{H}.T$ be the set of transitions of $\mathcal{H}$ starting at s and marked with g_i. The disjunction of the guards of T' is identically true.*
– *VI,VO,VM and VL are a partition of $\mathcal{H}.V$.*
– *For each $v_i \in VI$, the following two conditions hold:*
  - $\forall s \in \mathcal{H}.S : \quad \mathcal{H}.deriv(s) = extend(\mathcal{H}.deriv(s)|_{(\mathcal{H}.V \setminus \{v_i\})}, \mathcal{H}.V)$
  - $\forall t \in \mathcal{H}.T : \quad \forall a \in \mathcal{H}.guard(t) : \quad \forall u \in \mathcal{H}.update(t)(a) : \quad v_i \notin dom(u)$

  Note that the operation '$|$' is used as component-wise domain restriction for a set of functions.

Note. These definitions encapsulate our decisions for the difference between input signals and the other signals, and between input variables and the other variables. Note that output, multiply restricted and local signals and variables are not differentiated by these definitions. The differences between these concepts will be defined when we consider hybrid compositions.

The definition for input signals can be interpreted as follows: For each input signal and each location of the automaton, some transition labeled with the signal can always be taken. In this way the automaton does not restrict the input signal and thus it is not to blame for a deadlock.

The multiply restricted components are available for all access modes. A module as well as the environment for which a signal or variable is declared as multiply restricted can restrict the component in any way.

The definitions for input variables can be interpreted as follows:

– The derivation for an input variable may not be restricted in the 'deriv' set of any location of the automaton.
– The value of an input variable after a transition may not be restricted by one of the 'update' functions of a transition.

All variables which occur in an 'update' predicate must be declared as output, local or multiply restricted.

2.6 Compatibility of Hybrid Modules

To build a composition of hybrid modules we need some restrictions to ensure that the composition is a hybrid module again. For the definition of compatibility we need some notation.

Notation. Let $M = \{m_1, \ldots, m_n\}$ be a finite, nonempty set of hybrid modules ($|M| = n$). Let G be a finite set of signals with $G = GI \uplus GO \uplus GM \uplus GL$ (we use the symbol $\uplus$ for the disjoint union) and V be a finite set of variables ($V = VI \uplus VO \uplus VM \uplus VL$) for each module $m \in M$.

Let $G_{input} = \left(\bigcup_{m \in M} m.GI\right) - \left(\bigcup_{m \in M} m.GO \cup m.GM \cup m.GL\right)$ be the set of signals which are used at most as input signal in all the modules. These signals must not be restricted by the composed automaton.

If $\bar{t}$ is a tuple, then t_i denotes $\pi_i(\bar{t})$.

The set of all combinations of transitions (from modules of the set M) which are synchronized with input signal g is given by the following function:

$T : G_{input} \to 2^{(m_1.\mathcal{H}.T \times \cdots \times m_n.\mathcal{H}.T)}$, defined by:

$$T(g) =_{\text{def}} \left\{ \bar{t} \in m_1.\mathcal{H}.T \times \cdots \times m_n.\mathcal{H}.T \;\middle|\; \begin{array}{l} \exists k \in \{1,\ldots,n\} : m_k.\mathcal{H}.\text{sync}(t_k) = g \\ \wedge \left(\begin{array}{ll} \forall j \in \{1,\ldots,n\} : & (g \in m_j.G \wedge m_j.\mathcal{H}.\text{sync}(t_j) = g) \\ & \vee\ (g \notin m_j.G \wedge t_j \in m_j.\mathcal{H}.\text{noop}) \end{array} \right) \end{array} \right\}$$

Furthermore we need a function t to get the set of all tuples of transitions synchronized with g for a given signal g and a source location $\bar{s}$:

$t : G_{input} \times (m_1.\mathcal{H}.S \times \cdots \times m_n.\mathcal{H}.S) \to 2^{(m_1.\mathcal{H}.T \times \cdots \times m_n.\mathcal{H}.T)}$,
defined by:
$t(g, \bar{s}) =_{\text{def}} \left\{ \bar{t} \in T(g) \;\middle|\; \forall j \in \{1,\ldots,n\} : \pi_1(m_j.\mathcal{H}.\text{trans}(t_j)) = s_j \right\}$

Now we can define the compatibility of hybrid modules.

Definition 10. (Compatibility of sets of hybrid modules) *A nonempty, finite set of hybrid modules M is* **compatible** *if and only if there is a set G which is partitioned into four subsets GI, GO, GM and GL, the input, output, multiply restricted and locally defined signals of the hybrid composition, and a set V which is partitioned into four subsets VI, VO, VM and VL, the input, output, multiply restricted and locally defined variables of the hybrid composition, such that the following conditions hold:*

1. *Communication through common components:*
$$\forall m, m' \in M : m \neq m' \to (m.G \cap m'.G \subseteq G \wedge m.V \cap m'.V \subseteq V)$$
2. *Hiding of local components:*
$$\forall m, m' \in M : m \neq m' \to \begin{pmatrix} m.GL \cap m'.GL = \{\} = m.GL \cap G \\ \wedge\ m.VL \cap m'.VL = \{\} = m.VL \cap V \end{pmatrix}$$
3. *Usage of input components:*
$$\forall m \in M : \begin{pmatrix} GI \cap m.GO = \{\} = GI \cap m.GM \\ \wedge\ VI \cap m.VO = \{\} = VI \cap m.VM \end{pmatrix}$$
4. *Usage of output components:*
$$\forall m, m' \in M : m \neq m' \to \begin{pmatrix} m.GO \cap m'.G \subseteq m'.GI \\ \wedge\ m.GO \cap G \subseteq (GL \cup GO) \end{pmatrix}$$
$$\forall m, m' \in M : m \neq m' \to \begin{pmatrix} m.VO \cap m'.V \subseteq m'.VI \\ \wedge\ m.VO \cap V \subseteq (VL \cup VO) \end{pmatrix}$$
5. *Avoiding restriction of input signals:*
Let $V_{all} = \bigcup_{m \in M} m.V$ be the set of all variables.
$$\forall g \in G_{input} : \quad \forall \bar{s} \in m_1.\mathcal{H}.S \times \cdots \times m_n.\mathcal{H}.S :$$
$$\mathbb{R}^{V_{all}} = \bigcup_{\bar{t} \in t(g,\bar{s})} \left(\begin{array}{l} \bigcap_{1 \leq i \leq n} extend\,(m_i.\mathcal{H}.guard(t_i), V_{all}) \\ \cap \\ \left\{ a \in A\,(V_{all}) \;\middle|\; \begin{array}{l} \exists u \in V_{all} \nrightarrow \mathbb{R} : \\ \quad \forall i \in \{1,\ldots,n\} : \\ \quad\quad \exists u_i \in m_i.\mathcal{H}.update(t_i)(a|_{m_i.V}) : \\ \quad\quad\quad u_i = u|_{\text{dom}(u_i)} \end{array} \right\} \end{array} \right)$$

Note. In the following we explain these conditions:

1. Elements of M can at most communicate through elements of G and V. This means that for different hybrid modules in a hybrid composition, common signals and common variables must be elements of G and V.
2. The local signals and variables of different modules are different and the local signals and variables of a hybrid module are not used outside the hybrid module.
3. Input signals and input variables of the hybrid composition are not used as output or multiply restricted signals or variables in the component modules.
4. Output signals of a hybrid module may at most be used as input signals in the context of the hybrid module, and those of them which are members of the signals of the containing hybrid composition must be either locally defined or output signals. The analogue proposition is true for variables.
5. To avoid the restriction of input signals we have to ensure that at every point in time a transition is enabled to synchronize with an input signal. Thus we have to fulfill the condition that for each input signal in each location tuple the disjunction of the common guards is identically true and the set of 'update' functions does not forbid the transition. To construct this condition, we take the conjunction of the guards from all transitions in the common transition (note that the guards of no-op transitions are true) and then we have to subtract the guards for which a common update is not possible.

2.7 Hybrid Compositions

We define hybrid compositions as parallel composition of hybrid modules.

Definition 11. (Hybrid compositions) *A* **hybrid composition** *is a tuple* (G, V, M)*, where* G *is the finite set of signals,* V *is the finite set of variables and* M *is a compatible set of hybrid modules.*

The definitions for hybrid compositions contain some of the most important of the concepts we introduce in this work. They allow to structure a system into subsystems, and they contain the core of the concepts used to differentiate between different kinds of signals and variables.

What is missing is the possibility to define hierarchical systems. We allow this by defining a hybrid module which is equivalent to a given hybrid composition. The component modules in a hybrid composition can be combined to a hybrid automaton. This yields a hybrid module corresponding to the hybrid composition:

Definition 12. *Let $\mathcal{C}$ be a hybrid composition. The hybrid module described by the function hymod($\mathcal{C}$) corresponding to $\mathcal{C}$ is a flattened version of this hybrid composition $\mathcal{C}$. We define the function hymod$(\mathcal{C})$ in the following way:*

– $\mathit{hymod}(\mathcal{C}).GI =_{\mathit{def}} \mathcal{C}.GI$
– $\mathit{hymod}(\mathcal{C}).GO =_{\mathit{def}} \mathcal{C}.GO$
– $\mathit{hymod}(\mathcal{C}).GM =_{\mathit{def}} \mathcal{C}.GM$
– $\mathit{hymod}(\mathcal{C}).GL =_{\mathit{def}} \mathcal{C}.GL \cup (\bigcup_{m \in \mathcal{C}.M} m.G) - (\mathcal{C}.GI \cup \mathcal{C}.GO \cup \mathcal{C}.GM)$
– $\mathit{hymod}(\mathcal{C}).G =_{\mathit{def}} \mathit{hymod}(\mathcal{C}).GI \cup \mathit{hymod}(\mathcal{C}).GO \cup \mathit{hymod}(\mathcal{C}).GM \cup \mathit{hymod}(\mathcal{C}).GL$
– $\mathit{hymod}(\mathcal{C}).VI =_{\mathit{def}} \mathcal{C}.VI$
– $\mathit{hymod}(\mathcal{C}).VO =_{\mathit{def}} \mathcal{C}.VO$
– $\mathit{hymod}(\mathcal{C}).VM =_{\mathit{def}} \mathcal{C}.VM$
– $\mathit{hymod}(\mathcal{C}).VL =_{\mathit{def}} \mathcal{C}.VL \cup (\bigcup_{m \in \mathcal{C}.M} m.V) - (\mathcal{C}.VI \cup \mathcal{C}.VO \cup \mathcal{C}.VM)$
– $\mathit{hymod}(\mathcal{C}).V =_{\mathit{def}} \mathit{hymod}(\mathcal{C}).VI \cup \mathit{hymod}(\mathcal{C}).VO \cup \mathit{hymod}(\mathcal{C}).VM \cup \mathit{hymod}(\mathcal{C}).VL$
– $\mathit{hymod}(\mathcal{C}).\mathcal{H} =_{\mathit{def}} \prod_{m \in \mathcal{C}.M} m.\mathcal{H}$

Note. The local signals and variables of the hybrid module corresponding to a given hybrid composition consist of the local signals and variables of the hybrid composition itself and of all signals resp. variables of component modules which do not occur as interface signals resp. variables of the hybrid composition.

To get the product automaton for the whole composition we have to generate the product of all the sets of automata contained in the component modules.

Proposition 5. (hymod defines a hybrid module for a hybrid composition)
Let $\mathcal{C}$ be a hybrid composition. Then hymod$(\mathcal{C})$ is a hybrid module.

Proof. We have to show that the function $hymod(C)$ produces a hybrid automaton which fulfills the axioms in Definition 9. The first and the third axioms are fulfilled by the construction of the partitioned sets of signals and variables. The second axiom requires that the construction of the product automaton guarantees completeness of the guards for input signals. It is fulfilled by the compatibility of the hybrid modules in the composition: Item 7 of Definition 10, which ensures that no transition is avoided by an empty 'update' set, makes it possible to construct the product automaton $hymod(C).\mathcal{H}$. The fourth axiom requires that an input variable is restricted by neither the derivative function nor the 'update' set. This is fulfilled because if some 'deriv' sets (resp. some 'update' sets) do not restrict the input variable v then the composition of these sets also does not restrict v. Thus the constructed sets for derivations and value changes fulfill the conditions of Definition 9. □

2.8 Instantiation of Hybrid Modules

Often it is helpful to use several similar hybrid modules as components in a hybrid composition. For this, we will use instantiations of an existing hybrid module.

Definition 13. (Hybrid instantiations) *Let $\mathcal{M}$ and $\mathcal{M}'$ be hybrid modules with disjoint sets of signals and variables. A* **hybrid instantiation** *of module $\mathcal{M}$ for use in module $\mathcal{M}'$ consists of the following components:*

- *$\mathcal{M}$: The instantiated hybrid module.*
- *$\mathcal{M}'$: The context module in which $\mathcal{M}$ is instantiated.*
- *ident: An identification function assigning to each element of a subset of the signals and variables of $\mathcal{M}$ a signal or a variable of $\mathcal{M}'$. 'ident' has to fulfill the following conditions:*
  - *Signals are mapped to signals and variables are mapped to variables.*
  - *The function's domain does not contain local signals or variables of the instantiated hybrid module:*

$$dom(ident) \quad \subseteq \quad (\mathcal{M}.G - \mathcal{M}.GL) \cup (\mathcal{M}.V - \mathcal{M}.VL)$$

  - *The function identifies different signals and variables of the context module M' with different signals and variables of the instantiated hybrid module:*

$$\textit{ident is injective}$$

  - *Output signals and variables of the instantiated module may at most be identified with local or output signals resp. variables of the containing module:*

$$range(ident|_{\mathcal{M}.GO}) \subseteq \mathcal{M}'.GL \cup \mathcal{M}'.GO$$

$$range(ident|_{\mathcal{M}.VO}) \subseteq \mathcal{M}'.VL \cup \mathcal{M}'.VO$$

- *Multiply restricted signals and variables of the instantiated module may not be identified with input signals resp. input variables of the containing module:*

$$\mathit{range}(\mathit{ident}|_{\mathcal{M}.GM}) \cap \mathcal{M}'.GI = \{\}$$

$$\mathit{range}(\mathit{ident}|_{\mathcal{M}.VM}) \cap \mathcal{M}'.VI = \{\}$$

If $\mathcal{M}_1$ and $\mathcal{M}_2$ are both instantiated in a module M', different output signals of $\mathcal{M}_1$ and $\mathcal{M}_2$ may not be identified with the same signal in $\mathcal{M}'$, because this would mean that the signals are in fact multiply restricted. The same holds for variables. Outputs may only be identified with inputs of parallel modules. We encapsulate this observation in another definition.

Definition 14. (Consistency of sets of hybrid instantiations) *A set $s_{\mathcal{I}}$ of hybrid instantiations for which the context module is identical is said to be* **consistent** *if for different elements $\mathcal{I}_1$ and $\mathcal{I}_2$ of $s_{\mathcal{I}}$, the following two properties hold:*

$$\mathit{range}(\mathcal{I}_1.\mathit{ident}|_{\mathcal{I}_1.\mathcal{M}.GO}) \quad \cap \quad \mathit{range}(\mathcal{I}_2.\mathit{ident}|_{\mathcal{I}_2.\mathcal{M}.G}) \quad \subseteq \quad \mathit{range}(\mathcal{I}_2.\mathit{ident}|_{\mathcal{I}_2.\mathcal{M}.GI})$$

and

$$\mathit{range}(\mathcal{I}_1.\mathit{ident}|_{\mathcal{I}_1.\mathcal{M}.VO}) \quad \cap \quad \mathit{range}(\mathcal{I}_2.\mathit{ident}|_{\mathcal{I}_2.\mathcal{M}.V}) \quad \subseteq \quad \mathit{range}(\mathcal{I}_2.\mathit{ident}|_{\mathcal{I}_2.\mathcal{M}.VI})$$

Notation. (Renaming signals and variables of a hybrid automaton) Let $\mathcal{H}$ be a hybrid automaton and $\mathcal{M}$ be a hybrid module. Let the signals and variables of $\mathcal{H}$ and of $\mathcal{M}$ be disjoint. Let f be a function from a subset of $\mathcal{H}.G \cup \mathcal{H}.V$ to $\mathcal{M}.G \cup \mathcal{M}.V$.

We denote the **renaming of** $\mathcal{H}$ **by** f by $\mathcal{H}$.rename(f). This is the hybrid automaton resulting from $\mathcal{H}$ by replacing each signal and variable in the domain of f by the value f yields for it.

Note. Regarding Definition 10 (different modules have different local signals and variables) we do not need to rename the local signals and variables. Note that we speak about the signals and variables but not about their identification strings in the notation.

Definition 15. (The hybrid module for a hybrid instantiation) *Let $\mathcal{I}$ denote a hybrid instantiation. We define the hybrid module corresponding to $\mathcal{I}$, hymod$(\mathcal{I})$, in the following way:*

- $\mathit{hymod}(\mathcal{I}).GI =_{\mathit{def}} \mathit{range}(\mathcal{I}.\mathit{ident}|_{\mathcal{I}.\mathcal{M}.GI})$
- $\mathit{hymod}(\mathcal{I}).GO =_{\mathit{def}} \mathit{range}(\mathcal{I}.\mathit{ident}|_{\mathcal{I}.\mathcal{M}.GO})$
- $\mathit{hymod}(\mathcal{I}).GM =_{\mathit{def}} \mathit{range}(\mathcal{I}.\mathit{ident}|_{\mathcal{I}.\mathcal{M}.GM})$
- $\mathit{hymod}(\mathcal{I}).GL =_{\mathit{def}} \mathcal{I}.\mathcal{M}.G - (\mathcal{I}.\mathcal{M}.GI \quad \cup \quad \mathcal{I}.\mathcal{M}.GO \quad \cup \quad \mathcal{I}.\mathcal{M}.GM)$
- $\mathit{hymod}(\mathcal{I}).VI =_{\mathit{def}} \mathit{range}(\mathcal{I}.\mathit{ident}|_{\mathcal{I}.\mathcal{M}.VI})$
- $\mathit{hymod}(\mathcal{I}).VO =_{\mathit{def}} \mathit{range}(\mathcal{I}.\mathit{ident}|_{\mathcal{I}.\mathcal{M}.VO})$
- $\mathit{hymod}(\mathcal{I}).VM =_{\mathit{def}} \mathit{range}(\mathcal{I}.\mathit{ident}|_{\mathcal{I}.\mathcal{M}.VM})$
- $\mathit{hymod}(\mathcal{I}).VL =_{\mathit{def}} \mathcal{I}.\mathcal{M}.V - (\mathcal{I}.\mathcal{M}.VI \quad \cup \quad \mathcal{I}.\mathcal{M}.VO \quad \cup \quad \mathcal{I}.\mathcal{M}.VM)$
- $\mathit{hymod}(\mathcal{I}).\mathcal{H} =_{\mathit{def}} \mathcal{H}.\mathit{rename}(\mathcal{I}.\mathit{ident})$

Note. The function 'hymod' interprets hybrid instantiations as hybrid modules. With the help of this interpretation function, we can use consistent sets of hybrid instantiations as the components of a hybrid composition.

If the function 'ident' is not total then the variables and signals which are not in the domain of 'ident' are considered to be local.

Proposition 6. ('hymod' defines a hybrid module for a hybrid instantiation)
Let $\mathcal{I}$ denote a hybrid instantiation. Then hymod$(\mathcal{I})$ is a hybrid module.

Proof. Because 'ident' is injective, the sets GI, GO, GM, VI, VO and VM are constructed as disjoint sets. To the set of locals (GL and VL) we only add components which are not in the domain of 'ident'. The renaming of automata does not violate any condition of Definition 9. □

’hymod’ is the name of several functions yielding hybrid modules. One of them yields a hybrid module for a given hybrid composition, and another yields a hybrid module for a hybrid instantiation. With the help of these functions, we can use a hybrid instantiation or a hybrid composition in place of an explicitly given hybrid module wherever this is more convenient.

Another use of the hymod-functions is to fix semantics. We interpret hybrid compositions and hybrid instantiations in terms of hybrid modules. The hybrid automaton component of a hybrid module is interpreted as a labeled update system via the function ’us’. Thus, the only irreducible concept we introduced are the partition of the signals and variables of a hybrid automaton into input, output, multiply restricted and local components.

3 Conclusion

With respect to HyTech, Uppaal and Kronos, we introduced additional concepts into the formalism of hybrid automata. We claim that it helps to express some information explicitly in the formalism which is simple to grasp for a modeller and which can help to simplify formal analyses. It belongs to what we call “cheap and helpful redundancy”: Classification of signals and variables as input, output, multiply restricted and local should be easy for the modeller, and wrong suppositions about the use of a given signal or variable in one module can be checked syntactically by comparing its declaration with its use.

The CTA formalism is an extension of existing notations for modeling timed and hybrid systems. It extends the existing notations in order to better model different types of communication patterns several modules can use for interaction. Thus, we can for example express that a given variable or signal can never be restricted in a given module, which means that this module only reads this signal. Nevertheless, we can further use the synchronous semantics of CSP-like communication. One important consequence of the introduction of the new concepts is that it is now possible to explicitly specify that a given module only functions as an observer of a set of other modules. Other extensions with respect to existing notations allow to instantiate several times a module defined once, they introduce the usage of different name spaces for different modules, and they explain how interface components of an instantiated module are identified with components of the enclosing module.

For the newly introduced concepts, formal definitions have been given. These concepts fit well into the semantics of hybrid systems given as communicating automata which is used in Uppaal or HyTech, but they extend these concepts considerably with respect to more specific fixing of properties of the interface signals and variables, and for defining name spaces for different modules and their connection in instantiations. In Uppaal and HyTech all the variables are global and the set of automata has no hierarchical structure. We do not propose new algorithms, but we provide better support for modularly modeling large systems.

Another difference to HyTech is that in our semantics we support the following situation: One automaton sets a variable to some value such that the invariant of a location of another automaton becomes false. In HyTech this behavior leads to a deadlock and the modeller has to avoid such a situation. In a CTA, invariants have the meaning that no time can pass if the invariant is false, but the automaton with the false invariant can take a discrete transition and thus, the automaton can react on situations with false invariants.

We support our formalism with a tool performing automatical verification by reachability analysis (Beyer and Rust, 2000), but in the hybrid case only for systems for which the algorithm terminates (cf. (Alur *et al.*, 1997)). For the special class of Closed Timed Automata our tool Rabbit supports very efficient BDD-based reachability analysis (Beyer, 2001*c*). It uses an upper bound for the transition relation to compute good variable orderings (Beyer, 2001*b*). To allow also for modular proof strategies the tool provides refinement checking (Beyer, 2001*a*).

Acknowledgements

We thank Claus Lewerentz for discussions and comments on the work presented in this paper.

References

Alur, Rajeev and Thomas A. Henzinger (1996). Reactive modules. In: *Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science (LICS 1996)*. pp. 207–218.

Alur, Rajeev and Thomas A. Henzinger (1997). Modularity for timed and hybrid systems. In: *Proceedings of the 8th International Conference on Concurrency Theory (CONCUR'97)*. LNCS 1243. Springer-Verlag. Berlin. pp. 74–88.

Alur, Rajeev, Costas Courcoubetis and David Dill (1993). Model-checking in dense real-time. *Information and Computation* **104**, 2–34.

Alur, Rajeev, Thomas A. Henzinger and Howard Wong-Toi (1997). Symbolic analysis of hybrid systems. In: *Proceedings of the 36th International IEEE Conference on Decision and Control (CDC 1997)*.

Bengtsson, Johan, Kim Larsen, Fredrik Larsson, Paul Petersson and Wang Yi (1996). Uppaal – a tool suite for automatic verification of real-time systems. In: *Hybrid Systems III* (Rajeev Alur, Thomas A. Henzinger and Eduardo D. Sontag, Eds.). LNCS 1066. Springer-Verlag. Berlin. pp. 232–243.

Beyer, Dirk (2001*a*). Efficient reachability analysis and refinement checking of timed automata using BDDs. In: *Proceedings of the 11th Advanced Research Working Conference on Correct Hardware Design and Verifi cation Methods (CHARME 2001, Livingston)*. to appear, LNCS 2144. Springer-Verlag.

Beyer, Dirk (2001*b*). Improvements in BDD-based reachability analysis of timed automata. In: *Proceedings of the 10th International Symposium of Formal Methods Europe (FME 2001, Berlin): Formal Methods for Increasing Software Productivity* (Jose Nuno Oliveira and Pamela Zave, Eds.). LNCS 2021. Springer-Verlag. pp. 318–343.

Beyer, Dirk (2001*c*). Rabbit: Verification of real-time systems. Technical Report I-05/2001. BTU Cottbus.

Beyer, Dirk and Heinrich Rust (1998). Modeling a production cell as a distributed real-time system with Cottbus Timed Automata. In: *Tagungsband Formale Beschreibungstechniken für verteilte Systeme (FBT'98, Cottbus)* (Hartmut König and Peter Langendörfer, Eds.). Shaker Verlag, Aachen. pp. 148–159.

Beyer, Dirk and Heinrich Rust (2000). A tool for modular modelling and verification of hybrid systems. In: *Proceedings of the 25th IFAC/IFIP Workshop on Real-Time Programming 2000 (WRTP 2000, Palma)* (Alfons Crespo and Joan Vila, Eds.). Elsevier Science, Oxford. pp. 169–174.

Daws, C., A. Olivero, S. Tripakis and S. Yovine (1996). The tool KRONOS. In: *Hybrid Systems III* (Rajeev Alur, Thomas A. Henzinger and Eduardo D. Sontag, Eds.). LNCS 1066. Springer-Verlag. pp. 208–219.

Henzinger, Thomas A. (1996). The theory of hybrid automata. In: *Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science (LICS 1996)*. pp. 278–292.

Henzinger, Thomas A., Pei-Hsin Ho and Howard Wong-Toi (1995). A user guide to HyTech. In: *Proceedings of the 1st Workshop on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'95)*. LNCS 1019. Springer-Verlag. pp. 41–71.

Henzinger, Thomas A., Xavier Nicollin, Joseph Sifakis and Sergio Yovine (1994). Symbolic model-checking for real-time systems. *Information and Computation* **111**, 193–244.

Hoare, C.A.R. (1985). *Communicating Sequential Processes*. Prentice Hall. Hemel Hempstead.

Lamport, Leslie (1987). A fast mutual exclusion algorithm. *ACM Transactions on Computer Systems* **5**(1), 1–11.

Lynch, N., R. Segala, F. Vaandrager and H.B. Weinberg (1996). Hybrid I/O automata. In: *Hybrid Systems III* (Rajeev Alur, Thomas A. Henzinger and Eduardo D. Sontag, Eds.). LNCS 1066. Springer-Verlag. Berlin. pp. 496–510.

Lynch, Nancy A. and Mark R. Tuttle (1987). Hierarchical correctness proofs for distributed algorithms. In: *Proceedings of the 6th Annual ACM Symposium on Principles of Distributed Computing*. ACM. pp. 137–151.

Rust, Heinrich (1994). *Zuverlässigkeit und Verantwortung*. Vieweg. Braunschweig, Wiesbaden.