A Formalism for Modular Modelling of Hybrid Systems *

Dirk Beyer and Heinrich Rust**

Lehrstuhl Software-Systemtechnik, BTU Cottbus
Computer Science Reports 10/99, October 1999

Abstract. We present a formalism for modular modelling 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, but 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 modelling 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. [Rus94]).

In [BR98] a modelling notation is presented which allows to model hybrid systems in a modular way. It builds on the theoretical basis used in tools like UppAal [BLL+96], Kronos [DOTY96] and HyTech [HHWT95]. 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 (cf. [Hoa85]). Algorithms for these kinds of models have been presented in [ACD93] and [HNSY94].

In [BR98], we presented a refined notation which introduces the following concepts:

- 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.

---

* This paper is a revised version of a paper which appeared in the same report series [BR99]. Printing date: October 29, 1999.

** Reachable at: BTU, Postfach 101344, D-03013 Cottbus, Germany; Tel. +49(355)69-3803, Fax.:-3810; {db|rust}@informatik.tu-cottbus.de

- 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 state.
- Recurring subsystem components do not have 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 [LT87] and extended to hybrid systems in [LSVW96]. In [AH97], another approach is presented to describe modular hybrid systems. It builds on reactive modules [AH96] and extends them with continuously changing variables.

2 Formal definition

Our semantics for the basic underlying model is similar to that of [Hen96]. 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

The basic concept in our formalization is the hybrid automaton.

Definition 1. (Value assignments, configurations, hybrid automata) A **hybrid automaton** consists of the following components:

- S: A finite set of discrete states.
- G: A finite set of signals.
- V: A finite set of analogue variables.
  Variables are used in **value assignments**. A value assignment for a set of variables is a member from the set of functions $A(V) = V \to \mathbb{R}$.
  An element $c \in C = S \times A(V)$ is called **configuration**. It consists of a state of the automaton and a value assignment to its variables. A set of configurations is called a **region**.
- $I \subseteq C$: An initial condition, described as a set of configurations.
- T: A finite set of transitions.
- inv : $S \to 2^{A(V)}$: A function associating an invariant to each state. The invariant is a set of value assignments.
- deriv : $S \to 2^{A(V')}$: A function associating a set of admissible derivatives to each state. In the finite set of variables V' there is for each variable $v \in V$ the corresponding element v' which is used to define admissible time derivatives of the variable v. While the automaton remains in a state the continuous changes of a variable v are defined by their first time derivative v'.

- trans : $T \to S \times S$: A function associating a starting state and a target state to each transition.
- guard : $T \to 2^{A(V)}$: A function associating a guard with each transition.
- 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 sync(t) for transitions without a signal.
- allowed : $T \to (A(V) \to 2^{A(V)})$: A function associating with each transition a function which transforms a value assignment into one of a set of value assignments. The aim of this function is to restrict changes which may occur in any environment.
- initiated : $T \to (A(V) \to 2^{A(V)})$ with $\forall t \in T, a \in A(V)$: initiated(t)(a) $\subseteq$ allowed(t)(a): A function associating with each transition a function which transforms a value assignment into one of a set of value assignments. In contrast to 'allowed' the aim of this function is to restrict changes which occur without an environment.
  For each $t \in T$ and $a \in$ guard(t), the set initiated$(t)(a)$ must be nonempty. This condition ensures that the 'initiated' or 'allowed' 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:

- trans$(t_s) = (s, s)$
- guard$(t_s) =$ true
- sync$(t_s) = *$
- $\forall a \in A(V)$: initiated$(t_s)(a) = \{a\}$
- $\forall a \in A(V)$: allowed$(t_s)(a) = A(V)$

These transitions are no-op transitions. The subset of T consisting of all no-op transitions is referred to by 'noop'. The 'allowed' function of no-op transitions does not exclude any resulting value, and 'initiated' defines that no variable value changes. □

Notational convention: A typical case to use the 'initiated' predicate is to restrict variables which are not restricted by any parallel transition. If there is a variable which does not occur ticked in at least one inequation of the ALLOW clause, then this does not mean that the whole range of $\mathbb{R}$ is possible. For a variable which does not occur ticked in 'ALLOW' the meaning is that this transition does not change the value of the variable. Transitions of environmental automata are allowed to restrict the variable. But if no automaton restricts the variable x in its transition in the same point in time, then we use the information of the 'initiated' set which contains typically the additional restriction $x' = x$. To express that the whole range of $\mathbb{R}$ is possible for x after a transition, one might use the clause ALLOW $\{x' > 0$ OR $x' \leq 0\}$. In our notation we only use the typical case described above. Perhaps there are other useful aspects for a more general use of the 'initiated' set, but we did not yet find them, and thus we restrict our notation to have an easy to use syntax. Thus, in the INITIATE clause would be only restrictions of the form $x' = x$ for each variable $x \in X$ (set of variables known by the automaton), if x does not occur ticked in ALLOW. The consequence for us is to generate the 'initiated' set automatically, i. e. we do not have a syntactical clause for INITIATE in our notation.

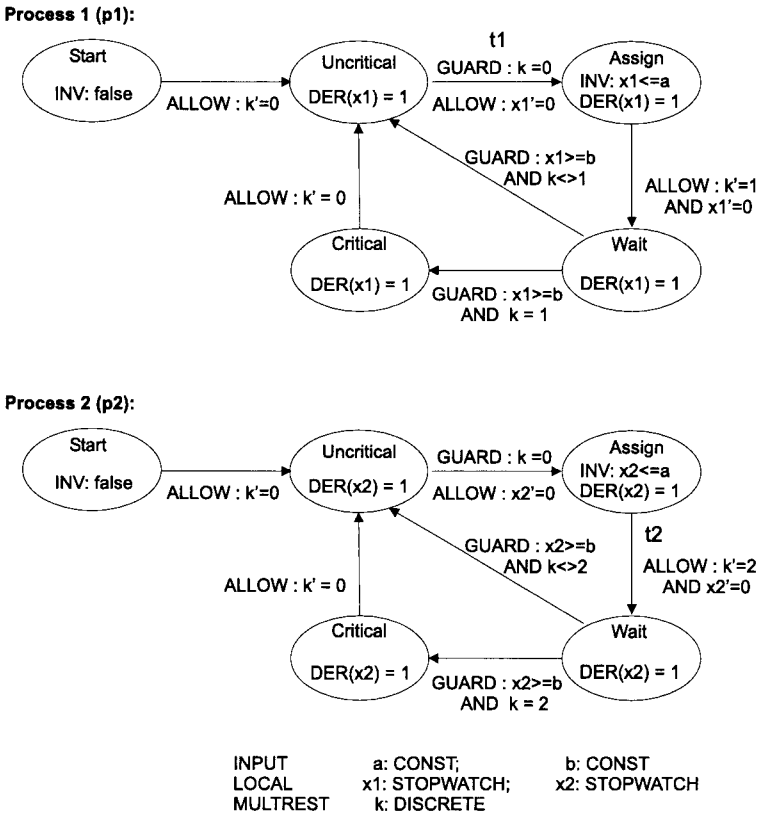

Fig. 1. Fischer's mutual exclusion protocol

Note. All variables which occur in an 'ALLOW' clause must be declared as OUTPUT, LOCAL, or MULTREST.

2.2 Illustration

To show the intention of the hybrid automata we display the automata for Fischer's timing-based mutual exclusion protocol in Fig. 1 (cf. [Lam87]). In our Fischer automaton a process is modelled by five states. 'Start' is the initial state for the automaton. From here it takes a transition initializing the shared variable k to the state which models the uncritical section. From this state 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 state modelling the 'Assign' statement. The clock x_i (with time derivation 1 in all states) measures the time staying in this state, and the invariant forces to leave the state after at most time a, which models the maximal time needed by the assign statement of the process. Then the transition to the 'Wait' state sets the variable k to the number of the process. In this state 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.

'Allowed' and 'Initiated'. Now we give an intention of our double transition predicates. The ALLOW clause at transition t_1 of process p_1 defines both the 'allowed' set and the 'initiated' set. The restriction for the 'allowed' set is $x_1' = 0$, which defines the new value of variable x_1 after the transition is taken. The 'initiated' set is additionally restricted by the notational convention given after Definition 1 above: $x_1' = 0$ AND $k' = k$. Thus variable k is not changed by this transition although k could be changed by a transition executed in parallel to this transition.

The advantage of the distinction beetween 'allowed' and 'initiated' is that we can express that the value of a variable is not changed by a transition in one automaton, but this transition allows a value change by a transition in a parallel automaton. If, for example, p_1 is in state 1 and it takes transition t_1 then we have two possible situations in process p_2:

- p_2 takes t_2 at the same point in time. The only valid value of variable k is 2 because of the ALLOW clause of t_2.
- If p_2 does not take t_2 in parallel, then the 'initiated' set of process p_1 forces $k' = 0$ (because of $k' = k$, and $k = 0$ by guard).

To present our notation we show the textual version of the system in Fig. 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 semantics of a hybrid automaton will be defined formally in the next section.

2.3 Labeled transition system semantics of hybrid automata

Notation. For a real u and two value assignments a and $a' \in A(V)$, let $u * a$ denote the function $\lambda(v : V) : u * a(v)$, and let $a + a'$ denote the funtion $\lambda(v : V) : a(v) + a'(v)$.

time : $C \times \mathbb{R} \times A(V) \to C$ is a function describing how the passage of some time u changes a configuration (s, a) when a time derivative $d \in A(V)$ for the variable values is fixed:

$$\text{time}((s, a), u, d) = (s, a + u * d)$$

□

A hybrid automaton can perform time transitions and discrete transitions.

Definition 2. (Time transitions and discrete transitions of a hybrid automaton) Let $\mathcal{H}$ be a hybrid automaton.

time($\mathcal{H}$) is the set of **time transitions of** $\mathcal{H}$. It is defined as the following set:

$$\begin{array}{l}
\{ \quad ((s, a_1), (s, a_2)) \in C \times C \\
\quad \mid \exists d \in \text{deriv}(s), u \in \mathbb{R}, u > 0 : \\
\qquad (\quad (s, a_2) = \text{time}((s, a_1), u, d) \\
\qquad \wedge \quad \forall (u' : 0 \leq u' \leq u) : \quad \text{time}((s, a_1), u', d) \in \text{inv}(s) \\
\qquad) \\
\}
\end{array}$$

```
1
2  MODULE Process {
3    // Constants for time bounds.
4    INPUT
5      // a is the maximal time the modelled assignment k:=1 needs.
6      a:           CONST;
7      // b is the minimal time the process waits for assignments
8      // initiated from other processes.
9      b:           CONST;
10     processNo:   CONST;  // Parameter for significant value of k.
11   MULTREST
12     k:  DISCRETE;        // k is the flag for announcement.
13   LOCAL
14     x:  CLOCK;           // A clock to measure the time in a state.
15
16   INITIALIZATION { STATE(Fisher) = start; }
17
18   AUTOMATON Fisher {
19     STATE start      { INV { FALSE; }
20                        TRANS uncritical { ALLOW { k'= 0; } } }
21     STATE uncritical { DERIV { DER(x) = 1;}
22                        TRANS assign     { GUARD { k  = 0; }
23                                           ALLOW { x'= 0; } } }
24     STATE assign     { INV { x <= a; }
25                        DERIV { DER(x) = 1; }
26                        TRANS wait       { ALLOW { x'= 0 AND
27                                                   k'= processNo; } } }
28     STATE wait       { DERIV { DER(x) = 1; }
29                        TRANS uncritical { GUARD { x >= b AND
30                                                   k <>processNo; } }
31                        TRANS critical   { GUARD { x >= b AND
32                                                   k = processNo; } } }
33     STATE critical   { DERIV { DER(x) = 1; }
34                        TRANS uncritical { ALLOW { k'= 0; } } }
35   }
36 }
37
38 MODULE System {
39   LOCAL
40     a = 3: CONST;      b = 3: CONST;
41     pNo1 = 1: CONST;   pNo2 = 2: CONST;
42   MULTREST
43     k:     DISCRETE;
44
45   INST Process1 FROM Process WITH {
46     a         AS a;
47     b         AS b;
48     processNo AS pNo1;
49     k         AS k;
50   }
51   INST Process2 FROM Process WITH {
52     a         AS a;
53     b         AS b;
54     processNo AS pNo2;
55     k         AS k;
56   }
57 }
58
```

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

discrete($\mathcal{H}$) is the set of **discrete transitions of** $\mathcal{H}$. It is defined as the following set:

$$
\begin{array}{l}
\{ \quad ((s_1, v_1), (s_2, v_2)) \\
\quad \mid \exists (t : T) : \\
\qquad (\quad \text{trans}(t) = (s_1, s_2) \\
\qquad \wedge \quad v_1 \in \text{guard}(t) \\
\qquad \wedge \quad v_2 \in \text{initiated}(t)(v_1) \\
\qquad) \\
\}
\end{array}
$$

□

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

Note. For discrete transitions the invariant is irrelevant. An invariant which is identically false can be used to construct urgent states, i. e. states in which time cannot pass.

The state component of the configuration may not change in a time transition.

In the definition of the set of discrete transitions, the signals and the 'allowed' function are not used. Their meaning is defined later, when we consider the parallel composition of automata. □

We will define a transition system semantics for hybrid automata.

Definition 3. (Transition system) A **transition system** consists of the following components:

- A (possibly infinite) set S of states, with a subset S_0 of initial states.
- A set $T \subseteq S \times S$ of transitions.

□

Notation. We use the point notation $A.x$ to address the component x of A. □

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

Definition 4. Let $\mathcal{H}$ be a hybrid automaton. The **transition system** ts($\mathcal{H}$) corresponding to $\mathcal{H}$ is defined in the following way:

- ts($\mathcal{H}$).$S =_{\text{def}} \mathcal{H}.C$.
- ts($\mathcal{H}$).$S_0 =_{\text{def}} \mathcal{H}.I$.
- ts($\mathcal{H}$).$T =_{\text{def}}$ time($\mathcal{H}$) $\cup$ discrete($\mathcal{H}$).

□

Note. The state space of the transition system consists of the configurations of the hybrid automaton. The set of starting states in the transition system is defined via the inital condition of the hybrid automaton. The transitions of the transition system are all time transitions and all discrete transitions of the transition system. □

2.4 Parallel composition of hybrid automata

In this section, we define what it means for two hybrid automata to be composed parallely.

We define the extension of a value assignment or a configuration by not restricting the values for the added variables, and some further notation.

Notation. For a tuple of length n, π_i with $i \in \{1, \ldots, n\}$ is a projection operator: it selects the i'th element.

For a function f, we write $\mathrm{dom}(f)$ for its domain and $\mathrm{ran}(f)$ for its range.

For a function f with domain D and with $D' \subseteq D$, we write $f\lceil D'$ for the restriction of f to D'. For a set F of functions, $F\lceil D'$ is the result of applying $\cdot\lceil D'$ componentwise.

Let $C \subseteq S \times A(V)$ be a set of configurations over the states 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 $\mathrm{extend}(B, V')$ and $\mathrm{extend}(C, V')$, are defined as follows:

$$\mathrm{extend}(B, V') =_{\mathrm{def}} \{a' \in A(V') | \exists a \in B : a = (a'\lceil V)\}$$
$$\mathrm{extend}(C, V') =_{\mathrm{def}} \{(s', a') \in S \times A(V') | \exists (s, a) \in C : s = s' \wedge a = (a'\lceil V)\}$$

□

Modular specification makes it necessary to combine several hybrid automata.

Definition 5. (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. [Hen96] for the general principle):

- $\mathcal{P}.S =_{\mathrm{def}} \mathcal{H}.\mathcal{S} \times \mathcal{H}'.\mathcal{S}$
- $\mathcal{P}.G =_{\mathrm{def}} \mathcal{H}.\mathcal{G} \cup \mathcal{H}'.\mathcal{G}$
- $\mathcal{P}.V =_{\mathrm{def}} \mathcal{H}.\mathcal{V} \cup \mathcal{H}'.\mathcal{V}$
- $\mathcal{P}.I =_{\mathrm{def}} \mathrm{extend}(\mathcal{H}.I, \mathcal{P}.V) \cap \mathrm{extend}(\mathcal{H}'.I, \mathcal{P}.V)$
- $\mathcal{P}.T =_{\mathrm{def}}$

$$\begin{array}{l} \{ \quad (t, t') \in \mathcal{H}.T \times \mathcal{H}'.T \\ \quad | \quad \mathcal{H}.\mathrm{sync}(t) \in \mathcal{H}'.G \rightarrow \mathcal{H}.\mathrm{sync}(t) = \mathcal{H}'.\mathrm{sync}(t') \\ \quad \wedge \quad \mathcal{H}'.\mathrm{sync}(t') \in \mathcal{H}.G \rightarrow \mathcal{H}'.\mathrm{sync}(t') = \mathcal{H}.\mathrm{sync}(t) \\ \quad \wedge \quad \mathcal{H}.\mathrm{sync}(t) \neq \mathcal{H}'.\mathrm{sync}(t') \rightarrow (t \in \mathcal{H}.\mathrm{noop} \vee t' \in \mathcal{H}'.\mathrm{noop}) \\ \} \end{array}$$

- $\mathcal{P}.\mathrm{inv}((s, s')) =_{\mathrm{def}} \mathrm{extend}(\mathcal{H}.\mathrm{inv}(s), \mathcal{P}.V) \cap \mathrm{extend}(\mathcal{H}'.\mathrm{inv}(s'), \mathcal{P}.V)$
- $\mathcal{P}.\mathrm{deriv}((s, s')) =_{\mathrm{def}} \mathrm{extend}(\mathcal{H}.\mathrm{deriv}(s), \mathcal{P}.V) \cap \mathrm{extend}(\mathcal{H}'.\mathrm{deriv}(s'), \mathcal{P}.V)$

- $\mathcal{P}.\text{trans}((t,t')) =_{\text{def}}$

$$\begin{array}{ll} (& (\pi_1(\mathcal{H}.\text{trans}(t)), \pi_1(\mathcal{H}'.\text{trans}(t'))) , \\ & (\pi_2(\mathcal{H}.\text{trans}(t)), \pi_2(\mathcal{H}'.\text{trans}(t'))) \\ ) & \end{array}$$

- $\mathcal{P}.\text{sync}((t,t')) =_{\text{def}}$

$$\begin{array}{ll} \text{if} & \mathcal{H}.\text{sync}(t) = * \\ \text{then} & \mathcal{H}'.\text{sync}(t') \\ \text{else} & \mathcal{H}.\text{sync}(t) \end{array}$$

- $\mathcal{P}.\text{allowed}((t,t'))(a) =_{\text{def}}$

$$\begin{array}{l} \text{extend}(\mathcal{H}.\text{allowed}(t)(a\lceil\mathcal{H}.V), \mathcal{P}.V) \\ \cap\, \text{extend}(\mathcal{H}'.\text{allowed}(t')(a\lceil\mathcal{H}'.V), \mathcal{P}.V) \end{array}$$

- For the definition of 'initiated' we need to refer the set of variables which are not restricted in the corresponding 'ALLOW' clause. We call this set 'Unmentioned'. $\mathcal{P}.\text{initiated}((t,t'))(a) =_{\text{def}}$

$$\begin{array}{ll} & \mathcal{P}.\text{allowed}((t,t'))(a) \\ \cap & \bigcap_{v \in Unmentioned} \text{extend}(a\lceil\{v\}, \mathcal{P}.V) \end{array}$$

- $\mathcal{P}.\text{guard}((t,t')) =_{\text{def}}$

$$\begin{array}{l} \text{extend}(\mathcal{H}.\text{guard}(t), \mathcal{P}.V) \\ \cap\, \text{extend}(\mathcal{H}'.\text{guard}(t'), \mathcal{P}.V) \\ \cap\, \{a \in A(\mathcal{P}.V) \quad | \quad \mathcal{P}.\text{allowed}((t,t'))(a) \neq \{\}\} \end{array}$$

□

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 intersection of the allowed-sets for a value assignment is empty or nonempty. In this way, the restriction for initiated-sets and allowed-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 allowed-sets.

The parallel composition allows all transformations of value assignments which are allowed by $\mathcal{H}$.allowed and $\mathcal{H}'$.allowed. If we do not restrict a variable in the 'allowed'-set of a transition, this means that we do not restrict the value of this variable after the transition has been taken. If no other automaton restricts this variable, it may get any value from the whole range of $\mathbb{R}$. But we often want to express that any change may happen to the variable during the transition, but that the given automaton itself on its own would not change the value. The solution to this problem is: If no automaton restricts a variable v and we would like to express that its value does not change, we set a restriction in 'initiated' that v has the same value as before the transition. □

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

Proposition 6. *Let $\mathcal{H}$ and $\mathcal{H}'$ be two hybrid automata. Then $\mathcal{P}$ 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 behaviour they are isomorphic.

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 modules.

Definition 7. (Hybrid modules) 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.
- $GMR \subseteq \mathcal{H}.G$: 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.
- $VMR \subseteq \mathcal{H}.V$: 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,GMR 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,VMR and VL are a partition of $\mathcal{H}.V$.
- For each $v_i \in VI$, the following two conditions hold:
  - For each $s \in \mathcal{H}.S$:

$$\mathcal{H}.\text{deriv}(s) = \\ \text{extend}(\mathcal{H}.\text{deriv}(s) \lceil (\mathcal{H}.V - \{v_i\}), \mathcal{H}.V)$$

- For each $t \in \mathcal{H}.T$ and each $a \in \mathcal{H}.\text{guard}(t)$:

$$\mathcal{H}.\text{allowed}(t)(a) = \\ \quad \text{extend}(\mathcal{H}.\text{allowed}(t)(a)\lceil(\mathcal{H}.V - \{v_i\}), \mathcal{H}.V), \\ \forall a' \in \mathcal{H}.\text{initiated(t)(a)}: a'(v_i) = a(v_i)$$

Note that the operation '$\lceil$' is used in the last two formulas as componentwise 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 state 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 time 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 MR 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 state of the automaton.
- The value of an input variable after a transition may not be restricted in the allowed-set of any value assignment in a transition.
- In difference to 'allowed', input variables can be restricted in 'initiated' to reflect that the value of the input variable is not changed by this automaton.

□

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 $G = GI \uplus GO \uplus GMR \uplus GL$ (we use the symbol $\uplus$ for the disjoint union) and V be a finite set of variables ($V = VI \uplus VO \uplus VMR \uplus VL$) for each module $m \in M$.

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

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:

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

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

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 starting state $\bar{s}$:

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

Now we can define the compatibility of hybrid modules.

Definition 8. (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, GMR 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, VMR 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:

$$\forall m, m' \in M : m \neq m' \rightarrow (m.G \cap m'.G \subseteq G \wedge m.V \cap m'.V \subseteq V)$$

2. Hiding of local components:

$$\forall m, m' \in M : m \neq m' \rightarrow \left(\wedge \begin{array}{l} m.GL \cap m'.GL = \{\} = m.GL \cap G \\ m.VL \cap m'.VL = \{\} = m.VL \cap V \end{array} \right)$$

3. Usage of input components:

$$\forall m \in M : \left(\wedge \begin{array}{l} GI \cap m.GO = \{\} = GI \cap m.GMR \\ VI \cap m.VO = \{\} = VI \cap m.VMR \end{array} \right)$$

4. Usage of output components:

$$\forall m, m' \in M : m \neq m' \rightarrow \left(\wedge \begin{array}{l} m.GO \cap m'.G \subseteq m'.GI \\ m.GO \cap G \subseteq (GL \cup GO) \end{array} \right)$$

$$\forall m, m' \in M : m \neq m' \rightarrow \left(\wedge \begin{array}{l} m.VO \cap m'.V \subseteq m'.VI \\ m.VO \cap V \subseteq (VL \cup VO) \end{array} \right)$$

5. Avoiding restriction of input signals:
   Let $V_{all} = \bigcup_{m \in M} m.V$ be the set of all variables.

$$\forall g \in G_{input} : \quad \forall \bar{s} \in m_1.\mathcal{H}.S \times \ldots \times m_n.\mathcal{H}.S :$$

$$\bigcup_{\bar{t} \in t(g,\bar{s})} \left(\begin{array}{l} \bigcap_{1 \le i \le n} extend\,(m_i.\mathcal{H}.guard(t_i), V_{all}) \\ \cap \\ \left\{ \begin{array}{l} a \in A\,(V_{all}) \\ \\ \quad \Big| \quad \bigcap_{1 \le i \le n} extend\,(m_i.\mathcal{H}.allowed(t_i)(a\lceil m_i.V), V_{all}) \\ \\ \quad \neq \{\} \end{array} \right\} \end{array} \right)$$

$$= \mathbb{R}^{V_{all}}$$

□

Note. 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 state tuple the disjunction of the common guards is identically true and the 'allowed' set does not forbid any transition. To construct this condition, we take the conjunction of the guards from all transitions in the common transition (note that the guards from no-op transitions are true) and then we have to subtract the guards, for which a common assignment is not possible.

2.7 Hybrid compositions

We define hybrid compositions as parallel composition of hybrid modules.

Definition 9. (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 10. 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:

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

□

Note 11. 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 rsp. variables of component modules which do not occur as interface signals rsp. variables of the hybrid composition.

Note 12. 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 13. (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 outline. We have to show that the function *hymod*(C) produces a hybrid automaton which fulfills the axioms in Definition 7. 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 has to be stable according to the completness of the guards for input signals. It is fulfilled by the compatibility of the hybrid modules in the composition. Item 7 of Definition 8, which ensures that no transition is avoided by an empty allowed-intersection, makes it possible to construct the product automaton $hymod(C).\mathcal{H}$. The fourth axiom requires that no input variable

is restricted by neither the derivative function nor the allowed function. This is fulfilled because the intersection of some deriv sets (rsp. allowed sets), which do not restrict the input variable v, does not restrict v, too. For nonrestricted variables (i.e. those which do not occur ticked in the 'ALLOW' clause) the initiated set does not change the values. Thus the constructed sets for derivations, allowed and initiated sets fulfill the conditions in Definition 7. □

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 14. (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:

$$\begin{array}{c}\text{dom(ident)}\\ \subseteq\\ (\mathcal{M}.G - \mathcal{M}.GL) \cup (\mathcal{M}.V - \mathcal{M}.VL)\end{array}$$

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

$$\text{ident is injective}$$

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

$$\text{ran}(\text{ident}\lceil\mathcal{M}.GO) \subseteq \mathcal{M}'.GL \cup \mathcal{M}'.GO$$

$$\text{ran}(\text{ident}\lceil\mathcal{M}.VO) \subseteq \mathcal{M}'.VL \cup \mathcal{M}'.VO$$

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

$$\text{ran}(\text{ident}\lceil\mathcal{M}.GMR) \cap \mathcal{M}'.GI = \{\}$$

$$\text{ran}(\text{ident}\lceil\mathcal{M}.VMR) \cap \mathcal{M}'.VI = \{\}$$

□

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 15. (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:

$$\begin{array}{l} \quad \text{ran}(\mathcal{I}_1.\text{ident}\lceil\mathcal{I}_1.\mathcal{M}.GO) \\ \cap \\ \quad \text{ran}(\mathcal{I}_2.\text{ident}\lceil\mathcal{I}_2.\mathcal{M}.G) \\ \subseteq \\ \text{ran}(\mathcal{I}_2.\text{ident}\lceil\mathcal{I}_2.\mathcal{M}.GI) \end{array}$$

and

$$\begin{array}{l} \quad \text{ran}(\mathcal{I}_1.\text{ident}\lceil\mathcal{I}_1.\mathcal{M}.VO) \\ \cap \\ \quad \text{ran}(\mathcal{I}_2.\text{ident}\lceil\mathcal{I}_2.\mathcal{M}.V) \\ \subseteq \\ \text{ran}(\mathcal{I}_2.\text{ident}\lceil\mathcal{I}_2.\mathcal{M}.VI) \end{array}$$

□

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 disjunct. 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 8 (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 16. (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:

- hymod($\mathcal{I}$).GI $=_{\text{def}}$ ran($\mathcal{I}$.ident$\lceil\mathcal{I}.\mathcal{M}.GI$)
- hymod($\mathcal{I}$).GO $=_{\text{def}}$ ran($\mathcal{I}$.ident$\lceil\mathcal{I}.\mathcal{M}.GO$)
- hymod($\mathcal{I}$).GMR $=_{\text{def}}$ ran($\mathcal{I}$.ident$\lceil\mathcal{I}.\mathcal{M}.GMR$)
- hymod($\mathcal{I}$).GL $=_{\text{def}}$ $\mathcal{I}.\mathcal{M}.G - (\mathcal{I}.\mathcal{M}.GI \cup \mathcal{I}.\mathcal{M}.GO \cup \mathcal{I}.\mathcal{M}.GMR)$
- hymod($\mathcal{I}$).VI $=_{\text{def}}$ ran($\mathcal{I}$.ident$\lceil\mathcal{I}.\mathcal{M}.VI$)
- hymod($\mathcal{I}$).VO $=_{\text{def}}$ ran($\mathcal{I}$.ident$\lceil\mathcal{I}.\mathcal{M}.VO$)
- hymod($\mathcal{I}$).VMR $=_{\text{def}}$ ran($\mathcal{I}$.ident$\lceil\mathcal{I}.\mathcal{M}.VMR$)
- hymod($\mathcal{I}$).VL $=_{\text{def}}$ $\mathcal{I}.\mathcal{M}.V - (\mathcal{I}.\mathcal{M}.VI \cup \mathcal{I}.\mathcal{M}.VO \cup \mathcal{I}.\mathcal{M}.VMR)$
- hymod($\mathcal{I}$).$\mathcal{H}$ $=_{\text{def}}$ $\mathcal{H}$.rename($\mathcal{I}$.ident)

□

Note 17. The function hymod interpretes 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. □

Note 18. If the function 'ident' is not total then the variables and signals which are not in the domain of 'ident' should be local.

Proposition 19. (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 outline. Because of the injectivity of 'ident' the sets GI, GO, GMR, VI, VO and VMR 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 7. □

'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 interprete hybrid compositions and hybrid instantiations in terms of hybrid modules. The hybrid automaton component of a hybrid module is interpreted as a transition system via the function ts. Thus, the only irreducible concepts we introduce are the partition of the signals and variables of a hybrid automaton into input, output, multiply restricted and local components.

3 Discussion

With respect to HyTech, UppAal and Kronos, we introduced additional concepts into the formalism of hybrid automata. This has to be justified somehow. 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 modelling 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 have new algorithms, but we have a better support for modularly modelling 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 state 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 reachability analysis and implementation checks on it, but only for systems for which the algorithm terminates (cf. [AHWT97]).

Acknowledgements

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

References

[ACD93] Rajeev Alur, Costas Courcoubetis, and David Dill. Model-checking in dense real-time. *Information and Computation*, 104:2–34, 1993.

[AH96] Rajeev Alur and Thomas A. Henzinger. Reactive modules. In *Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science (LICS 1996)*, pages 207–218, 1996.

[AH97] Rajeev Alur and Thomas A. Henzinger. Modularity for timed and hybrid systems. In *Proceedings of the 8th International Conference on Concurrency Theory (CONCUR'97)*, LNCS 1243, pages 74–88, Berlin, 1997. Springer-Verlag.

[AHWT97] Rajeev Alur, Thomas A. Henzinger, and Howard Wong-Toi. Symbolic analysis of hybrid systems. In *Proceedings of the 36th International IEEE Conference on Decision and Control (CDC 1997)*, 1997.

[BLL+96] Johan Bengtsson, Kim Larsen, Fredrik Larsson, Paul Petersson, and Wang Yi. Uppaal – a tool suite for automatic verification of real-time systems. In Rajeev Alur, Thomas A. Henzinger, and Eduardo D. Sontag, editors, *Hybrid Systems III*, LNCS 1066, pages 232–243, Berlin, 1996. Springer-Verlag.

[BR98] Dirk Beyer and Heinrich Rust. Modeling a production cell as a distributed real-time system with cottbus timed automata. In Hartmut König and Peter Langendörfer, editors, *FBT'98: Formale Beschreibungstechniken für verteilte Systeme*, pages 148–159, June 1998.

[BR99] Dirk Beyer and Heinrich Rust. A modular hybrid modelling notation. Technical Report I-3/1999, BTU Cottbus, 1999.

[DOTY96] C. Daws, A. Olivero, S. Tripakis, and S. Yovine. The tool KRONOS. In Rajeev Alur, Thomas A. Henzinger, and Eduardo D. Sontag, editors, *Hybrid Systems III*, LNCS 1066, pages 208–219, Berlin, 1996. Springer-Verlag.

[Hen96] Thomas A. Henzinger. The theory of hybrid automata. In *Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science (LICS 1996)*, pages 278–292, 1996.

[HHWT95] Thomas A. Henzinger, Pei-Hsin Ho, and Howard Wong-Toi. A user guide to HyTech. In *Proceedings of the First Workshop on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, LNCS 1019, pages 41–71. Springer-Verlag, 1995.

[HNSY94] Thomas A. Henzinger, Xavier Nicollin, Joseph Sifakis, and Sergio Yovine. Symbolic model-checking for real-time systems. *Information and Computation*, 111:193–244, 1994.

[Hoa85] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice Hall, Hemel Hempstead, 1985.

[Lam87] Leslie Lamport. A fast mutual exclusion algorithm. *ACM Transactions on Computer Systems*, 5(1):1–11, 1987.

[LSVW96] N. Lynch, R. Segala, F. Vaandrager, and H.B. Weinberg. Hybrid I/O automata. In Rajeev Alur, Thomas A. Henzinger, and Eduardo D. Sontag, editors, *Hybrid Systems III*, LNCS 1066, pages 496–510, Berlin, 1996. Springer-Verlag.

[LT87] Nancy A. Lynch and Mark R. Tuttle. Hierarchical correctness proofs for distributed algorithms. In *Proceedings of the 6th Annual ACM Symposium on Principles of Distributed Computing*, pages 137–151. ACM, August 1987.

[Rus94] Heinrich Rust. *Zuverlässigkeit und Verantwortung*. Vieweg, Braunschweig, Wiesbaden, 1994.