Preprint from Proceedings of MBMV-Workshop, Oldenburg, Germany, February 2011

Fully Symbolic Model Checking for Timed Automata

Georges Morbé and Christoph Scholl
Albert-Ludwigs-Universität Freiburg, Institut für Informatik,
D-79110 Freiburg im Breisgau, Germany
Email:{morbe, scholl}@informatik.uni-freiburg.de

Abstract

In this paper we introduce a new formal model, called finite state machines with time (FSMT), to represent real-time systems. We present a model checking algorithm for FSMTs, which works on fully symbolic state sets containing both the clock values and the state variables. In order to verify timed automata (TA) with our model checking algorithm, we present two different methods to convert TAs to FSMTs. In addition to pure interleaving semantics we can convert TAs to FSMTs having a parallelized interleaving behavior which allows parallelism of transitions causing no conflicts. This can dramatically reduce the number of steps during verification. Our experimental results show that our prototype implementation outperforms the state of the art model checker uppaal.

I. Introduction

The application area of real-time systems grows with an enormous speed and along with that grows the complexity and the damage caused by the failure of such systems. Therefore, verification of such systems becomes more and more important. In this paper we introduce a new formal model for real-time systems, called finite state machines with time (FSMT) which is well-suited for symbolic verification algorithms. We present a fully symbolic model checking algorithm for FSMTs. In order to verify timed automata (TA) [1], [2], [4] we present a method to convert a TA into an FSMT. In addition to normal interleaving semantics of TAs we can give a symbolic representation of an FSMT simulating a 'parallelized interleaving' behavior, which allows parallelism of transitions causing no conflicts. This parallelized interleaving behavior can dramatically reduce the number of steps during verification.

Today's state of the art model checkers for TAs, like uppaal [3], [9] work on a semi-symbolic representation of the state space, the so-called difference bound matrices (DBM). These model checkers explicitly run through all discrete locations of the TA while maintaining a symbolic representation of the reached clock values. This approach has a high performance on small TAs but cannot handle the enormous amount of discrete states in large systems. Our model checking algorithm uses LinAIGs [7], [6] to describe the state space. Where DBMs combine a symbolic representation for the clock values with an explicit representation of locations, LinAIGs can represent both the continuous part (i.e. the clock values) and the discrete part (i.e. the state variables) symbolically. This allows us to handle larger systems of TAs than model checking tools using a semi-symbolic representation.

First experimental results show that our prototype implementation outperforms uppaal in both configurations, for pure interleaving behavior and for parallelized interleaving behavior. The results also indicate that for benchmarks allowing parallelized interleaving behavior this approach has a stunning performance due to reduction of the steps during verification.

The paper is organized as follows. In Sect. II we give a brief review of the well-known timed automata (TA), then we introduce the finite state machines with time (FSMT) in Sect. III. In Sect. IV we provide an insight into the functioning of our model checking algorithm. In Sect. V we introduce a method to convert a TA into an FSMT. Sect. VI is dedicated to the results where we evaluate our approach with both configurations. We conclude the paper in Sect. VII.

II. Preliminaries – Timed Automata

Real-time systems are often represented as timed automata (TA) [1], [2], [4].

TAs use clock variables $X := \{x_1, \ldots, x_n\}$. The set of clock constraints $\mathcal{C}(X)$ contains atomic constraints of the form $(x_i \sim n)$ and $(x_i - x_j \sim n)$ with $n \in \mathbb{Q}$ and $\sim \in \{<, \leq, =, \geq, >\}$. Let $\mathcal{C}_c(X)$ be the set of conjunctions over clock constraints. $c \in \mathcal{C}_c(X)$ describes a subset of $(\mathbb{R}_0^+)^n$, namely the set of all valuations of variables in X which evaluate c to true.

We consider TAs with integer variables. Let $Int := \{int_1, \ldots, int_r\}$ be a set of bounded integer variables. $lb : Int \to \mathbb{Z}$ and $ub : Int \to \mathbb{Z}$ assign lower and upper bound to $int_i \in Int$ ($lb(int_i) \leq ub(int_i)$). Let $Assign(Int)$ be the set of assignments to integer variables. The right-hand side of an

assignment to an integer variable int_i may be an integer arithmetic expression over integer variables and integer constants. Let $Cond\,(Int)$ be a set of constraints of the form $(int_i \sim n)$ and $(int_i \sim int_j)$ with $n \in \mathbb{Z}$, $\sim \in \{<, \leq, =, \geq, >\}$ and $int_i, int_j \in Int$.

Example 1 *The timed automaton TA_i shown in Fig. 1 has only one clock variable, $X = \{x_i\}$. It has three 'locations' $s_0^{(i)}$, $s_1^{(i)}$, and $s_2^{(i)}$. Locations are connected by transitions which may be labeled. The transition $\left(s_2^{(i)} \rightarrow s_0^{(i)}\right)$, e.g., is labeled with the guard $int = i$, with an assignment $int := 0$, and with the clock reset $x_i := 0$. The location $s_1^{(i)}$ is labeled with a clock constraint $x_i \leq 5$ which is a so-called location invariant.*

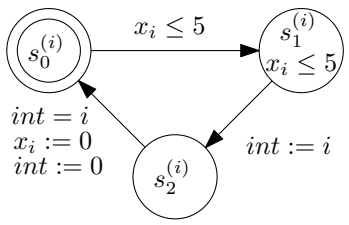


Fig. 1. Timed automaton TA_i

In general, transitions in TAs are labeled with guards, actions, assignments to integers and resets of clocks. Guards are restricted to conjunctions of clock constraints and constraints on integers. Actions from $Act := \{a_1, \ldots, a_k\}$ are used for synchronization between different TAs. For our purposes they do not have a special meaning when considering one timed automaton in isolation. Transitions in different automata labeled with the same actions are taken simultaneously. If a transition in a TA is not labeled by an action, then this transition can only be taken, if all other TAs stay in their current location. Resets are assignments to clock variables of the form $x_i := 0$. Invariants in TAs are conjunctions of clock constraints assigned to locations. A TA may stay in a location as long as the location invariant is not violated. Timed automata are formally defined as follows:

Definition 1 (Timed Automaton) *A timed automaton (TA) is a tuple $\langle L, l_0, X, Act, Int, lb, ub, E, Inv\rangle$ where L is a finite set of locations, $l_0 \in L$ is an initial location, $X = \{x_1, \ldots, x_n\}$ is a finite set of real-valued clock variables, Act is a finite set of actions, $Int = \{int_1, \ldots, int_r\}$ is a finite set of integer variables. $lb : Int \rightarrow \mathbb{Z}$ and $ub : Int \rightarrow \mathbb{Z}$ assign lower and upper bound to each $int_i \in Int$ with $lb(int_i) \leq ub(int_i)$ for $1 \leq i \leq r$, $E \subseteq L \times (\mathcal{C}(X) \cup Cond\,(Int)) \times (Act \cup \{\epsilon\}) \times 2^X \times 2^{Assign(Int)} \times L$ is a set of transitions and the function $Inv : L \rightarrow \mathcal{C}_c(X)$ assigns a conjunction of clock constraints as invariant to each location. If for $e = (l, g_e, act, r_e, assign_e, l') \in E$ it holds that $act \in Act$, then we call e a transition with a synchronizing action; if $act = \epsilon$, then we call e a transition without synchronizing action.*

Definition 2 (Semantics of a Timed Automaton) *Let $TA = \langle L, l_0, X, Act, Int, lb, ub, E, Inv\rangle$ be a timed automaton. A state of TA is a combination of a location and a valuation of the clock variables and integer variables.*

- *There is a continuous transition from state $s = (l, x_1^v, \ldots, x_n^v, int_1^v, \ldots, int_r^v)$ to state $s' = (l, x_1^w, \ldots, x_n^w, int_1^v, \ldots, int_r^v)$ $(s \rightarrow^c s')$ iff $(x_1^v, \ldots, x_n^v)$ and $(x_1^w, \ldots, x_n^w)$ fulfill $Inv(l)$, $lb(int_i) \leq int_i^v \leq ub(int_i)$ $\forall 1 \leq i \leq r$, and there is $t \in \mathbb{R}_0^+$ with $\forall 1 \leq j \leq n : x_j^w = x_j^v + t$.*
- *There is a discrete transition from state $s = (l, x_1^v, \ldots, x_n^v, int_1^v, \ldots, int_r^v)$ to state $s' = (l', x_1^w, \ldots, x_n^w, int_1^w, \ldots, int_r^w)$ $(s \rightarrow^d s')$ iff $(x_1^v, \ldots, x_n^v)$ fulfills $Inv(l)$, $(x_1^w, \ldots, x_n^w)$ fulfills $Inv(l')$, $lb(int_i) \leq int_i^v \leq ub(int_i)$, $lb(int_i) \leq int_i^w \leq ub(int_i)$ $\forall 1 \leq i \leq r$, and $\exists e = (l, g_e, act, r_e, assign_e, l') \in E$ with $(x_1^v, \ldots, x_n^v, int_1^v, \ldots, int_r^v)$ fulfills the guard g_e, $x_i^w = 0$ if $x_i \in r_e$ otherwise $x_i^w = x_i^v$, the values $int_1^w, \ldots, int_r^w$ result from $int_1^v, \ldots, int_r^v$ by applying the assignments in $assign_e$.*
- *$\rightarrow = \rightarrow^d \cup \rightarrow^c$ is the transition relation of TA. A trajectory of TA is a finite or infinite sequence of states $(s^j)_{j \geq 0}$ with $s^0 = (l_0, 0, \ldots, 0, lb(int_1), \ldots, lb(int_r))$ and $s^{j-1} \rightarrow s^j$ for each $j > 0$. A state is reachable, if there is a trajectory ending in that state.*

A timed system is a system of p timed automata $\{TA_1, \ldots, TA_p\}$. A timed system has an interleaving semantics, i.e., transitions in different timed automata may not be taken simultaneously unless they synchronize over actions. For simplicity, we assume that only two timed automata are able to synchronize over a binary synchronization channel, i.e., we restrict ourselves to timed systems where an action may only synchronize two different TAs. The composition of p timed automata is again a timed automaton:

Definition 3 *Let $TA_1, \ldots, TA_p$ be a timed system with $TA_i = \left\langle L^{(i)}, l_0^{(i)}, X, Act, Int, lb, ub, E^{(i)}, Inv^{(i)}\right\rangle$. Let $A(act) = \{TA_i \mid \exists e = (l, g_e, act, r_e, assign_e, l') \in E^{(i)}\}$ for each $act \in Act$. We assume that*

$|A(act)| \leq 2$ *for each* $act \in Act$. *The composition of* $TA_1, \ldots, TA_p$ *is* $TA = \left\langle \left(L^{(1)} \times \ldots \times L^{(p)}\right), \left(l_0^{(1)}, \ldots, l_0^{(p)}\right), X, Act, Int, lb, ub, E, Inv \right\rangle$ *where* E *is the smallest set with the following property:*

- *If for* $1 \leq i \leq p$ $\exists e = (l_i, g_e, act, r_e, assign_e, l_i') \in E^{(i)}$, $act = \epsilon$ *or* $|A(act)| = 1$, *then* $((l_1, \ldots, l_i, \ldots, l_p),\ g_e, act, r_e, assign_e, (l_1, \ldots, l_i', \ldots, l_p)) \in E$.
- *If for* $1 \leq i, j \leq p$ *with* $i \neq j$: $\exists e_i = (l_i, g_{e_i}, act, r_{e_i},\ assign_{e_i}, l_i') \in E^{(i)}$, $\exists e_j = (l_j, g_{e_j}, act, r_{e_j},\ assign_{e_j},\ l_j') \in E^{(j)}$, $act \in Act$, *then* $((l_1, \ldots, l_i, \ldots, l_j, \ldots, l_p), g_{e_i} \wedge g_{e_j}, act, r_{e_i} \cup r_{e_j}, assign_{e_i} \cup assign_{e_j}, (l_1, \ldots, l_i',\ \ldots, l_j', \ldots, l_p)) \in E$.

$Inv(l_1, \ldots, l_p) := \bigwedge_{i=1}^{p} Inv^{(i)}(l_i)$ *for all* $(l_1, \ldots, l_p) \in L^{(1)} \times \ldots \times L^{(p)}$.

III. Finite State Machines with Time

Now we present a new formal model to represent real-time systems, the finite state machines with time, which are especially suited for being represented symbolically. A finite state machine with time, to which we will refer as FSMT in this paper, is an extension of finite state machines by real-valued clock variables used to represent time. Later on, we will present a fully symbolic model checking algorithm for FSMTs and then a translation from TAs into FSMTs.

Let $X := \{x_1, \ldots, x_n\}$ be the set of real-valued clock variables, $Y := \{y_1, \ldots, y_l\}$ a set of (binary) state variables, $I := \{i_1, \ldots, i_h\}$ a set of (binary) input variables. Let $\mathcal{C}_b(X)$ be the set of arbitrary boolean combinations of clock constraints and $\mathcal{C}_b(X, Y)$ be the set of arbitrary boolean combinations of clock constraints and state variables (similarly for $\mathcal{C}_b(X, Y, I)$). As usual, $c \in \mathcal{C}_b(X, Y)$ describes a subset of $(\mathbb{R}_0^+)^n \times \{0,1\}^l$, namely the set of all valuations of variables in X and Y which evaluate c to true. An FSMT is defined as follows (see Fig.2 for an illustration):

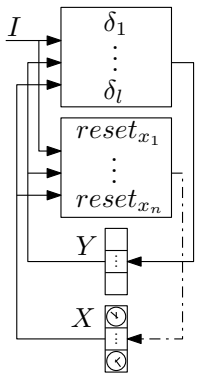


Fig. 2. FSM with time

Definition 4 (FSMT) *A finite state machine with time (FSMT) is a tuple* $\langle X, Y, I, init, (\delta_1, \ldots, \delta_l), (reset_{x_1}, \ldots, reset_{x_n}), Inv \rangle$ *where* $X := \{x_1, \ldots, x_n\}$ *is a set of clock variables,* $Y := \{y_1, \ldots, y_l\}$ *is a set of state variables,* $I := \{i_1, \ldots, i_h\}$ *is a set of input variables,* $init : (\mathbb{R}_0^+)^n \times \{0,1\}^l \to \{0,1\}$ *is a predicate describing the set of initial states,* $\delta_i : (\mathbb{R}_0^+)^n \times \{0,1\}^l \times \{0,1\}^h \to \{0,1\}$ $(1 \leq i \leq l)$ *are transition functions,* $reset_{x_j} : (\mathbb{R}_0^+)^n \times \{0,1\}^l \times \{0,1\}^h \to \{0,1\}$ $(1 \leq j \leq n)$ *are reset functions,* $Inv : (\mathbb{R}_0^+)^n \times \{0,1\}^l \to \{0,1\}$ *is a predicate describing a state invariant, and* $init \wedge \neg Inv = 0$. *The functions* δ_i *and* $reset_{x_j}$ *can be represented by boolean combinations from* $\mathcal{C}_b(X, Y, I)$, $init$ *and* Inv *can be represented by boolean combinations from* $\mathcal{C}_b(X, Y)$.

A state of an FSMT is a valuation $s = (x_1^v, \ldots, x_n^v, y_1^v, \ldots, y_l^v) \in (\mathbb{R}_0^+)^n \times \{0,1\}^l$ of the clock variables and the state variables. A valuation $(y_1^v, \ldots, y_l^v)$ is also called a *location* of the FSMT. Trajectories of an FSMT always start in states fulfilling $init$ and all states on these trajectories have to fulfill the state invariant Inv. An FSMT may perform discrete steps which are defined by transition functions δ_i based on the valuations of clocks, state variables, and inputs. When performing a discrete step, a clock x_i is reset to 0 iff $reset_{x_i}$ evaluates to 1. Moreover an FSMT may perform continuous steps (or time steps) where it stays in the same location, but lets time pass. This means that all clocks may be increased by the same constant as long as the resulting state stays in the set described by Inv. More formally, the semantics of FMSTs is defined as follows:

Definition 5 (Semantics of an FSMT)
Let $F = \langle X, Y, I, init, (\delta_1, \ldots, \delta_l), (reset_{x_1}, \ldots, reset_{x_n}), Inv \rangle$ *be an FSMT.*

- *There is a discrete transition from state* $s = (x_1^v, \ldots, x_n^v, y_1^v, \ldots, y_l^v)$ *to state* $s' = (x_1^w, \ldots, x_n^w, y_1^w, \ldots, y_l^w)$ $(s \to^d s')$ *iff* $Inv(s) = Inv(s') = 1$ *and there is* $(i_1^v, \ldots, i_h^v) \in \{0,1\}^h$ *with*
  - $\forall 1 \leq i \leq l : y_i^w = \delta_i(x_1^v, \ldots, x_n^v, y_1^v, \ldots, y_l^v, i_1^v, \ldots, i_h^v)$,
  - $\forall 1 \leq j \leq n : x_j^w = \begin{cases} x_j^v, & \text{if } reset_{x_j}(x_1^v, \ldots, x_n^v, y_1^v, \ldots, y_l^v, i_1^v, \ldots, i_h^v) = 0 \\ 0, & \text{if } reset_{x_j}(x_1^v, \ldots, x_n^v, y_1^v, \ldots, y_l^v, i_1^v, \ldots, i_h^v) = 1, \end{cases}$

- *There is a continuous transition from state* $s = (x_1^v, \ldots, x_n^v, y_1^v, \ldots, y_l^v)$ *to state* $s' = (x_1^w, \ldots, x_n^w, y_1^w, \ldots, y_l^w)$ ($s \to^c s'$) *iff* $Inv(s) = Inv(s') = 1$, $\forall 1 \leq i \leq l : y_i^w = y_i^v$, *and there is* $t \in \mathbb{R}_0^+$ *with* $\forall 1 \leq j \leq n : x_j^w = x_j^n + t$.[1]
- $\to = \to^d \cup \to^c$ *is the transition relation of* F. *A trajectory of* F *is a finite or infinite sequence of states* $(s^j)_{j \geq 0}$ *with* $init(s^0) = 1$ *and* $s^{j-1} \to s^j$ *for each* $j > 0$. *A state is reachable, if there is a trajectory ending in that state.*

We consider systems of FSMTs $\{F_1, \ldots, F_p\}$, where the components are running in parallel. Communication in such a system is realized just as for communicating FSMs. FSMTs communicate by reading each other's state variables, clocks, and shared input variables. Thus, composition of FSMTs is done just be replacing input variables of the components by state variables of other components or by inputs of the overall system. The composition of p FSMTs $F_1, \ldots, F_p$ is again an FSMT:

Definition 6 *Let* $F_1, \ldots, F_p$ *be FSMTs with* $F_i = \left\langle X, Y^{(i)}, I^{(i)}, init^{(i)}, \delta^{(i)}, (reset_{x_1}^{(i)}, \ldots, reset_{x_n}^{(i)}), Inv^{(i)} \right\rangle$, $Y^{(i)} = \{y_1^{(i)}, \ldots, y_{l_i}^{(i)}\}$, $I^{(i)} = \{i_1^{(i)}, \ldots, i_{h_i}^{(i)}\}$. *Let* $map : \bigcup_{i=1}^p I^{(i)} \to (I \cup \bigcup_{i=1}^p Y^{(i)})$ *be a mapping for the inputs of components* $F_1, \ldots, F_p$. *Then the composition of* $F_1, \ldots, F_p$ *wrt.* map *is an FSMT* F *with*

$$F = \left\langle X, \bigcup_{i=1}^{p} Y^{(i)}, I, \bigwedge_{i=1}^{p} init^{(i)}, \left(\tilde{\delta}^{(1)}, \ldots, \tilde{\delta}^{(p)}\right), \left(\bigvee_{i=1}^{p} reset_{x_1}^{(i)}, \ldots, \bigvee_{i=1}^{p} reset_{x_n}^{(i)}\right), \bigwedge_{i=1}^{p} Inv^{(i)} \right\rangle,$$

$\tilde{\delta}^{(i)}(x_1, \ldots, x_n, y_1^{(i)}, \ldots, y_{l_i}^{(i)}, i_1, \ldots, i_h) = \delta^{(i)}(x_1, \ldots, x_n, y_1^{(i)}, \ldots, y_{l_i}^{(i)}, map(i_1^{(i)}), \ldots, map(i_{h_i}^{(i)}))$, *and* $I = \{i_1, \ldots, i_h\}$.

IV. Model checking algorithm

Our model checking algorithm works on an FSMT $F = \langle X, Y, I, init, (\delta_1, \ldots, \delta_l), (reset_{x_1}, \ldots, reset_{x_n}), Inv \rangle$ as defined in Def. 4. It checks whether a state not fulfilling a safety predicate $safe \in \mathcal{C}_b(X, Y)$ is reachable from some initial state fulfilling $init$. The algorithm works backwards, i.e. it starts with the state set representation $\neg safe$ and – step by step – it computes sets of states from which $\neg safe$ can be reached. After each step it checks whether the newly reached states include some initial states. If this is the case, the safety property is not fulfilled. If the fixpoint iteration converges and the backward traversal does not reach any initial state, then the safety property holds. For computing all backwards reachable states we alternate between continuous and discrete steps.

A. Continuous step

Let $\Phi(x_1, \ldots, x_n, y_1, \ldots, y_l)$ be a state set of our model checking algorithm. Then the state set after a continuous step (letting time pass) is $\Phi'(x_1, \ldots, x_n, y_1, \ldots, y_l)$.

$$\Phi'(x_1, \ldots, x_n, y_1, \ldots, y_l) = \left[\bigwedge_{i=1}^{n} (x_i \geq 0)\right] \wedge [\exists \lambda (\lambda \geq 0) \Phi(x_1 + \lambda, \ldots, x_n + \lambda, y_1, \ldots, y_l)] \wedge Inv(x_1, \ldots, x_n, y_1, \ldots, y_l) \quad (1)$$

We refer to the result Φ' of the backwards continuous step as the continuous preimage $Pre_c(\Phi)$.

B. Discrete step

In our model checking algorithm the discrete step is computed from the state set $\Phi' = Pre_c(\Phi)$. The resulting state set $Pre_d(\Phi')$ contains all predecessors of Φ' from which Φ' can be reached by a discrete transition in the FSMT. The first part of the discrete step is a substitution of the state variables and the clock constraints in the current state set representation Φ'. (Note that as an invariant of our model checking algorithm all computed state set representations are in $\mathcal{C}_b(X, Y)$, i.e., they are boolean combinations of boolean variables and clock constraints.) Each state variable y_i is substituted with its transition function δ_i:

$$y_i \leftarrow \delta_i(x_1, \ldots, x_n, y_1, \ldots, y_l, i_1, \ldots, i_h) \quad (2)$$

[1] Usually we require that Inv has the following property: If we fix variables $y_1, \ldots, y_l$ of Inv to arbitrary constant values 0 or 1, then the resulting predicate shall describe a convex set. If this would not be the case, then there could be a continuous transition from s to s' with time step of length t, but no continuous transition from s to s'' with time step of length $t' < t$, since $Inv(s'') = 0$.

Consider a clock constraint of the form $(x_i - x_j \sim n)$ with $x_i, x_j \in X$, $\sim \in \{<, \leq, =, \geq, >\}$ and $n \in \mathbb{Q}$. There are only four possible cases how a clock constraint can be changed due to resets executed during a transition: (1) x_i and x_j are reset, (2) only x_i is reset, (3) only x_j is reset or (4) none of the clock variables in the constraint is reset. We use the reset conditions $reset_{x_i}$ to determine when a clock variable x_i is reset. The substitution for each clock constraint of the form $(x_i - x_j \sim n)$ in the state set is then

$$\begin{aligned}(x_i - x_j \sim n) \leftarrow (\ (\ reset_{x_i} \wedge reset_{x_j} \wedge (0 \sim n)\ ) \vee (\ \overline{reset_{x_i}} \wedge reset_{x_j} \wedge (x_i \sim n)\ ) \vee \\ (\ reset_{x_i} \wedge \overline{reset_{x_j}} \wedge (x_j \sim n)\ ) \vee (\ \overline{reset_{x_i}} \wedge \overline{reset_{x_j}} \wedge (x_i - x_j \sim n)\ )\ )\end{aligned} \quad (3)$$

(Of course, $(0 \sim n)$ reduces to constant 0 or 1.)

$\Phi''(x_1, \ldots, x_n, y_1, \ldots, y_l, i_1, \ldots, i_h)$ is obtained from $\Phi'(x_1, \ldots, x_n, y_1, \ldots, y_l)$ by substituting all state variables as shown in formula (2) and all clock constraints as shown in formula (3) simultaneously.

The second part of the discrete step is a quantification of the boolean input variables $i_1, \ldots, i_h$ in Φ'' followed by an intersection with the invariant Inv.

$$\begin{aligned}\Phi'''(x_1, \ldots, x_n, y_1, \ldots, y_l) = \left[\bigwedge_{i=1}^{n} (x_i \geq 0)\right] \wedge [\exists i_1, \ldots, i_k\ \Phi''(x_1, \ldots, x_n, y_1, \ldots, y_l, i_1, \ldots, i_h)] \\ \wedge Inv(x_1, \ldots, x_n, y_1, \ldots, y_l)\end{aligned} \quad (4)$$

The resulting state space Φ''' holds all states from which Φ' can be reached by performing a discrete transition in the FSMT. We refer to Φ''' as the discrete preimage $Pre_d(\Phi')$.

C. Algorithm

Our model checking algorithm is a backwards model checking algorithm, starting with $\neg safe$ as mentioned above. The main loop consists of a continuous step defined in Sect. IV-A and a discrete step defined in Sect. IV-B. After each of these steps we test whether one of the initial states was reached. The main loop is left when an initial state was reached (which means that the safety property is violated) or when a fixpoint is reached (which means that the safety property holds).

Algorithm 1 Model checking algorithm

$\Phi_0 := \neg safe; \Phi_{collect} := 0; i := 0$
while $(\Phi_i \wedge \neg\Phi_{collect} \neq 0)$ **do**
  if $(\Phi_i \wedge \text{init} \neq 0)$ **then return** *false*
  $\Phi_{collect} := \Phi_{collect} \vee \Phi_i$
  $i := i + 1$
  $\Phi_i := Pre_c(\Phi_{i-1})$
  if $(\Phi_i \wedge \text{init} \neq 0)$ **then return** *false*
  $\Phi_i := Pre_d(\Phi_i)$
return *true*

D. Implementation

We implemented a prototype of the model checking algorithm using LinAIGs [7], [6], [12] for representing sets of states. LinAIGs are able to provide a compact representation for arbitrary boolean combinations of linear constraints and boolean variables (which of course include the formulas from $\mathcal{C}_b(X, Y)$). LinAIGs contain both a boolean and a continuous part. The boolean part of LinAIGs is represented by functionally reduced AIGs (FRAIGs) [10], [11], which are boolean circuits consisting only of and gates and inverters. 'Functionally reduced' means that every node in the FRAIG represents a unique boolean function. In order to represent the continuous part, LinAIGs use a set of boolean constraint variables Q where each linear constraint is encoded by some $q_l \in Q$. For keeping the overall representation as compact as possible LinAIGs make heavy use of SMT solvers [5], [8]. LinAIGs support quantification of boolean and real variables which makes them a very powerful data structure which fits exactly the technical needs of our implementation. Moreover, since in our application the linear constraints are restricted to clock constraints, we do not need SMT solvers for full linear arithmetic, but only for difference logic which can be solved much more efficiently. This makes the LinAIG approach even more effective.

V. From Timed Automata to FSMTs

In order to be able to verify systems of TAs using our framework presented so far, we present how to convert a system of TAs into an FSMT.

Components of FSMTs run in parallel, whereas components of TAs run asynchronously (one after the other) according to the interleaving semantics (unless parallelism is enforced by synchronization actions). In our translation we consider two different implementations of the interleaving semantics of TAs. At first, in Sect. V-B, we show how to transform a TA into an FSMT keeping its *pure interleaving* behavior. Then, in Sect. V-C, we present how to convert a TA into an FSMT with a *parallelized interleaving*

behavior, in which we allow – in addition to single steps of components according to the interleaving semantics – parallelism for transitions causing no conflicts when taken in parallel. The different conflicts possible with parallelized interleaving behavior are also described in Sect. V-C. The motivation for the parallelized interleaving variant consists in an accelerated state space traversal.

A. First steps of translation

We consider a system of p timed automata $\{TA_1, \ldots, TA_p\}$.

The locations of timed automaton TA_q $(1 \leq q \leq p)$ are encoded with boolean variables $y_1^{(q)}, \ldots, y_{l_q}^{(q)}$ (the location bits) for which we use a logarithmic encoding with $l_q = \lceil log\,(L_q) \rceil$. The sets of location bits of two different TAs are disjoint.

The integer variable int_i with $(1 \leq i \leq r)$ occurring in the timed system is replaced by a binary encoding of boolean variables $b_1^{(i)}, \ldots, b_{f_i}^{(i)}$ (the integer bits). As $lb\,(int_i)$ and $ub\,(int_i)$ are known for all $(1 \leq i \leq r)$, the number of integer bits f_i needed to represent int_i is also known.

The location bits and the integer bits together form the set of state bits $\{y_1, \ldots, y_l\}$.

The location invariants in a TA can be merged into one condition for the complete automaton of the form $Inv^{(q)}\left(y_1^{(q)}, \ldots, y_{l_q}^{(q)}, x_1, \ldots, x_n\right)$ (by a simple conjunction of one implication for each location with the meaning 'if TA_q is in location l, then the location invariant of l holds').

A timed automaton TA_q with $(1 \leq q \leq p)$ has a total of $m_q := |E_q|$ transitions. Assume that transition i is a transition with the discrete location $\left(\epsilon_1^{(i,s)}, \ldots, \epsilon_{l_q}^{(i,s)}\right)$ as source and the discrete location $\left(\epsilon_1^{(i,d)}, \ldots, \epsilon_{l_q}^{(i,d)}\right)$ as destination. Let the transition i be labeled with a guard $g_i^{(q)}(x_1, \ldots, x_n)$ and a reset set $r_i^{(q)} \in 2^{\{x_1,\ldots,x_n\}}$. The guard $g_i^{(q)}$ is extended by the constraint that the source of its corresponding edge is location $\left(\epsilon_1^{(i,s)}, \ldots, \epsilon_{l_q}^{(i,s)}\right)$, i.e., it is changed to the new guard $g_i'^{(q)} := g_i^{(q)} \wedge \left((y_1^{(q)})^{\epsilon_1^{(i,s)}} \wedge \ldots \wedge (y_{l_q}^{(q)})^{\epsilon_{l_q}^{(i,s)}}\right)$.

Moreover, a transition i in TA_q may be labeled with a synchronization action $a_{q,i}$. How to treat these actions is shown in section V-B for interleaving behavior and in section V-C for parallelized interleaving behavior. The following modifications which have to be done to convert a TA into an FSMT depend on whether we want to have pure interleaving behavior or parallelized interleaving behavior.

B. Modifications for Pure Interleaving Behavior

In order to use the model checking algorithm with pure interleaving behavior, it has to be assured that at any time only one TA may take a transition while the others remain in their current location unless of course two TAs synchronize. We have two kind of transitions which have to be considered separately:

- For transitions without synchronization actions it has to be ensured that transitions of two different TAs are not enabled at the same time. For this we use new input variables $\{e_{l-1}, \ldots, e_0\}$, $l = \lceil log\,(p) \rceil$ in a system of p timed automata and we add different assignments for these new input variables to the guards of such transitions: For each transition i in a timed automaton TA_q which is not labeled with a synchronization action we add these input variables to the guard $g_i'^{(q)}$ and get a new guard $g_i''^{(q)} = g_i'^{(q)} \wedge \left(e_{l-1}^{q_{l-1}} \wedge \ldots \wedge e_0^{q_0}\right)$ with $bin\,(q) = (q_{l-1}, \ldots, q_0)$. $bin\,(q)$ is the binary representation of q.
- For transitions labeled with a synchronization action we cannot use the previous modification as this would cause the synchronized transitions not be enabled at the same time. Let us assume that transition i in TA_q and the transition j in TA_k are labeled with the same action $a_{\{(q,i),(k,j)\}}$. Then $A\left(a_{\{(q,i),(k,j)\}}\right) = \{TA_q, TA_k\}$ with $bin\,(k) = (k_{l-1}, \ldots, k_0)$ and $bin\,(q) = (q_{l-1}, \ldots, q_0)$. To assure synchronization without the use of actions we extend the guards of the synchronized transitions. The new guard of transition i in TA_q and of transition j in TA_k is $g_i''^{(q)} = g_j''^{(k)} := g_i'^{(q)} \wedge g_j'^{(k)} \wedge \left(\left(e_{l-1}^{q_{l-1}} \wedge \ldots \wedge e_0^{q_0}\right) \vee \left(e_{l-1}^{k_{l-1}} \wedge \ldots \wedge e_0^{k_0}\right)\right)$. This allows us to realize synchronization without using actions simply by the fact that one component may read the state bits and inputs of another component.

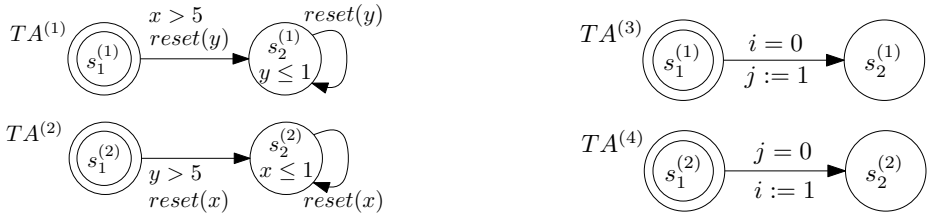
Since for an FSMT we have to define transition functions, we have to avoid the case that there is a state where no transition into a successor state is enabled. For this reason we introduce a self loop to every location in each timed automaton TA_q. The self loop of a location l_i gets as guard the conjunction

of the negated guards of all outgoing transitions, thus the self loop of a location is enabled whenever no other outgoing transition is enabled. Moreover, we have to exclude non-deterministic behavior (as allowed for TAs) to arrive at deterministic transition *functions* for FSMTs. When a transition in a TA is enabled, it has not to be taken; the automaton can choose to stay in the current location. Additionally, when more than one transition is enabled at the same time it is chosen non-deterministically which one is taken. To establish determinism in a TA we use new input variables. For a set of t transitions with the same source and non-disjoint guards we need $\lceil log\,(t) \rceil$ input variables to make the guards disjoint. These input variables can be shared within a TA but must not be shared among different TAs. A timed automaton TA_q requires $t^{(q)} = \left\lceil log\left(t^{(q)}_{max}\right)\right\rceil$ input variables to guarantee determinism, where $t^{(q)}_{max}$ is the maximum of non-disjoint transitions with the same source.

After these transformations we can build the transition functions, reset conditions and invariant to get an FSMT representation of the timed system with pure interleaving behavior. This is shown in section V-D.

C. Modifications for Parallelized Interleaving Behavior

In the previous section we have seen which modifications have to be done to convert a timed system into a system of FSMTs with pure interleaving behavior. In this section we will show what has to be done to get a system of FSMT with parallelized interleaving behavior. To this end several conflicts have to be solved.



(a) Read/write problem on clocks

(b) Read/write problem on integers

Fig. 3. Conflicts caused by parallel behavior

- In a parallelized interleaving run there may be conflicts caused by resets of clock variables. Consider the timed system shown in in Fig. 3(a) which consists of components $TA^{(1)}$ and $TA^{(2)}$. When parallel transitions of two components are allowed, the state $\left(s_2^{(1)}, s_2^{(2)}, 0, 0\right)$ is reachable from state $\left(s_1^{(1)}, s_1^{(2)}, 6, 6\right)$ by taking the transitions from $s_1^{(1)}$ and $s_1^{(2)}$ in parallel. But according to interleaving semantics this state is unreachable. If w.l.o.g. $TA^{(1)}$ takes the transition leaving its initial state first, then it resets the clock y and y will never be larger than 1. Thus the transition of $TA^{(2)}$ from $s_1^{(2)}$ will never be enabled and $TA^{(2)}$ always stays in its initial location. (A similar observation holds for the case that $TA^{(2)}$ is executed first.)
  To avoid the problem of reaching more states than allowed by the semantics of interleaving, we force the timed system to simulate a pure interleaving behavior in such cases by adding input variables. For each clock variable $x_t \in \{x_1, \ldots, x_n\}$ we use one new input variable i_t. The guard of a transition i in TA_q resetting the clock variable x_t is then enlarged to $g_i''^{(q)} = g_i'^{(q)} \wedge \neg i_t$ and the guard of a transition j in TA_k with a clock constraint over x_t is then $g_j''^{(k)} = g_j'^{(k)} \wedge i_t$. Thus no transition reading the value of clock x_t is enabled at the same time as a transition resetting x_t.
- Another conflict of the same type may occur with integers. It is obvious that two transitions updating the same integer int_i must not be taken in parallel because of write/write problems. But, just as we have seen for clock variables there may also be read/write conflicts on integer variables. In the timed system consisting of $TA^{(3)}$ and $TA^{(4)}$ shown in Fig. 3(b) the state $\left(s_2^{(1)}, s_2^{(2)}\right)$ is not reachable according to interleaving semantics. However it is reachable, if transitions can be taken in parallel. Just as for the read/write conflict for clock variables we force the timed system to take an interleaving behavior for transitions causing conflicts on integer variables. We introduce write-enable numbers for integers. Assume integer int_i is written in q timed automata $TA_{i_1}, \ldots, TA_{i_q}$, then the write-enable

number for int_i is:

$$we^{int_i} = \begin{cases} bin\,(0)\,, & \text{if } int_i \text{ is read in a guard of a transition,} \\ bin\,(1)\,, & \text{if } TA_{i_1} \text{ updates } int_i, \\ & \vdots \\ bin\,(q)\,, & \text{if } TA_{i_q} \text{ updates } int_i. \end{cases}$$

To encode we^{int_i} we need $\lceil log\,(q+1) \rceil$ input variables. The guard of each transition reading the value of integer int_i is extended by '$we^{int_i} = bin\,(0)$'. Each guard of a transition in TA_{i_k} $(1 \leq k \leq q)$ which updates int_i is extended by '$we^{int_i} = bin\,(k)$'. This makes it impossible that two TAs write int_i at the same time, since the corresponding guards cannot be enabled at the same time. Equally it is impossible that any integer variable is read and updated in the same discrete transition.

The synchronization is handled in a similar way as we have seen in Sect. V-B for pure interleaving behavior. We use the ability of an FSMT to read the state bits and inputs of another FSMT to force two transitions to be taken simultaneously. Lets assume that transition i in TA_q and transition j in TA_k are labeled with the same synchronization action $a_{\{(q,i),(k,j)\}}$. Then $A\left(a_{\{(q,i),(k,j)\}}\right) = \{TA_q, TA_k\}$, and the guards of both transitions are changed to $g_i^{\prime\prime(q)} = g_j^{\prime\prime(k)} := g_i^{\prime(q)} \wedge g_j^{\prime(k)}$. The action $a_{\{(q,i),(k,j)\}}$ is no longer needed to synchronize the transitions. Both components in the system synchronize by reading each others state bits and inputs.[2]

Parallelized interleaving is introduced to accelerate model checking runs by reaching certain states faster. But of course, we should not lose intermediate states of interleaved executions. For that reason we give each component the non-deterministic choice to stay in its current location during a discrete step. For this we introduce a self loop with guard 'true' to every location in the automaton. By taking this transition the automaton does not leave the current location and does no assignments to clocks or integer variables. Then, to introduce determinism we do the same modifications using input variables as we have done for pure interleaving behavior in Sect. V-B.

The resulting system is deterministic and has a parallelized interleaving behavior. In the following section we show how to compute transition functions, reset conditions and a global invariant.

D. Computation of a symbolic representation

Based on the guards $g_i^{\prime\prime(q)}$ for transitions i of TA_q (from $\left(\epsilon_1^{(i,s)}, \ldots, \epsilon_{l_q}^{(i,s)}\right)$ to $\left(\epsilon_1^{(i,d)}, \ldots, \epsilon_{l_q}^{(i,d)}\right)$) as computed in Sect. V-B or V-C it is easy to compute the transition functions for state bits encoding locations of TA_q. We have to consider $m_q^\prime$ transitions for T_q (including new self loops added in Sect. V-B or V-C). The transition function $\delta_j^{(q)}$ computes when the state bit j in the modified automaton TA_q is set to true. (Assume that the set of all input variables we have added according to Sect. V-B or V-C is $\{i_1, \ldots, i_h\}$.) Then

$$\delta_j^{(q)}\left(x_1, \ldots, x_n, y_1^{(q)}, \ldots, y_{l_q}^{(q)}, i_1, \ldots, i_h\right) = \bigvee_{\substack{1 \leq i \leq m_q^\prime \\ \epsilon_j^{(i,d)} = 1}} g_i^{\prime\prime(q)}\left(x_1, \ldots, x_n, y_1^{(q)}, \ldots, y_{l_q}^{(q)}, i_1, \ldots, i_h\right) \quad (5)$$

The transition functions for state bits resulting from encoding of integer variables are derived from location encodings, the guards computed in Sect. V-B or V-C, and right-hand side expressions of assignments.[3] Details are omitted here.

Besides the transition functions we need the reset functions for clocks. The following function indicates when the clock variable x_i is reset in TA_q:

$$reset_{x_i}^{(q)}\left(x_1, \ldots, x_n, y_1^{(q)}, \ldots, y_{l_q}^{(q)}, i_1, \ldots, i_h\right) = \bigvee_{\substack{1 \leq i \leq m_q^\prime \\ x_i \in r_i^{(q)}}} g_i^{\prime\prime(q)}\left(x_1, \ldots, x_n, y_1^{(q)}, \ldots, y_{l_q}^{(q)}, i_1, \ldots, i_h\right) \quad (6)$$

The overall reset function for clock x_i is then computed by $reset_{x_i} = \bigvee_{q=1}^{p} reset_{x_i}^{(q)}$.

[2]For ease of exposition we omit the special case of concurrent read / write or write / write on synchronizing transitions here.

[3]In our prototype implementation we restrict the right-hand sides of assignments to integer constants, integer variables and additions of two integers.

As a last component of the FSMT computed from timed automata $TA_1, \ldots, TA_P$, we compute the global invariant Inv simply by conjunction of all local invariants $Inv^{(q)}$.

The transition functions, reset conditions, and the Invariant provide a fully symbolic representation of the corresponding FSMT. Our model checking algorithm uses this representation to perform fully symbolic model checking.

VI. Experimental results

Table I shows the results of our prototype on three different benchmarks. We ran our prototype with pure interleaving behavior (FSMT MC interleaving) and with parallelized interleaving behavior (FSMT MC parallel) and compare the results to uppaal. We have conducted all experiments on a 16 core AMD Opteron with 1.2 Ghz and 64 GB RAM with a time limit of 3600 seconds and a memory limit to 2 GB.

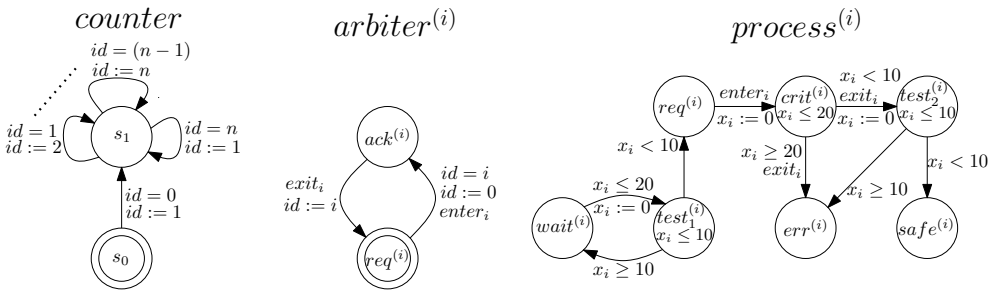


Fig. 4. Processes with arbiter

The first benchmark is a system consisting of n TAs $TA_1, \ldots, TA_n$ as shown in Fig. 1 with $\neg safe := \bigwedge_{i=1}^{n} s_1^{(i)}$ (which is reachable from the initial states). Comparing pure interleaving and parallelized interleaving, we can observe an enormous performance gain for parallelized interleaving due to a reduction of the number of steps in state space traversal. Our algorithm with parallelized interleaving behavior can finish state space traversal just after one step by taking the transition $\left(s_0^{(i)} \to s_1^{(i)}\right)$ for all i in parallel. Our algorithm with pure interleaving behavior computes in one step for each state reached so far all the predecessors reachable by one backwards step of an arbitrary automaton. Thus in this simple example it needs n steps for a system with n processes. Uppaal performs much worse on this example, since it works on an explicit representation of locations and it computes all possible permutations of enabled transitions step by step.

The second benchmark is the well known fischer protocol [13]. As we can see in Table I the results of our algorithm with a pure interleaving behavior are better than the results with a parallelized interleaving behavior. This is caused by the fact that the fischer protocol does not allow parallel behavior. Even if we run our model with a parallelized interleaving behavior, a pure interleaving behavior is simulated due to the write-enable numbers for the integer variable used in the benchmark. These additional inputs of the write-enable numbers which have to be quantified in the discrete step are responsible for the loss of performance. But in both configurations for pure interleaving and for parallelized interleaving behavior our symbolic model checking algorithm can solve systems with a lot more processes than uppaal.

The last benchmark is shown in Fig. 4. It consists of a counter for an integer id, a distributed arbiter with n components, and n processes. The arbiter guarantees that only one process can enter a critical region. The counter determines which process is allowed to enter this region and once a process i enters it, the arbiter component i blocks the counter so that no other process may take a transition to a critical location. $arbiter^{(i)}$ and $process^{(i)}$ communicate over the actions $enter_i$ and $exit_i$. The counter communicates with all arbiter components over the integer id.

The property which is verified in this benchmark is $\bigwedge_{i=1}^{n} crit^{(i)}$, which is not reachable. As we can see in Table I our model checking algorithm is able to handle much more processes than uppaal. As this benchmark allows parallel behavior our model checking algorithm with parallelized interleaving performs best and it can solve up to 17 processes where uppaal runs into a timeout of 3600 seconds already for 7 processes.

VII. Conclusions

We presented a new formal model to represent real-time systems, the finite state machine with time, which is well-suited for symbolic verification algorithms. We presented a backwards model checking

benchmark 1	UPPAAL breadth first	UPPAAL depth first	FSMT MC interleaving	FSMT MC parallel
12	20.85	59.76	59.35	0.49
13	83.33	217.78	113.27	0.67
14	324.94	792.26	195.94	0.57
15	memout	memout	765.02	0.68
16	memout	memout	1276.95	0.72
17	memout	memout	1973.31	1.47
18	memout	memout	2129.43	0.81
19	memout	memout	timeout	1.17
fischer protocol	UPPAAL breadth first	UPPAAL depth first	FSMT MC interleaving	FSMT MC parallel
10	17.09	57.49	65.97	129.54
11	70.91	312.1	137.7	291.7
12	283.52	1705.87	335.5	876.41
13	1122.5	timeout	671.22	1370.47
14	memout	timeout	1306.25	3536.29
15	memout	timeout	409.45	1410.59
16	memout	timeout	1299.75	1664.18
17	memout	timeout	1003.42	timeout
18	memout	timeout	1975.06	timeout
19	memout	timeout	2940.39	timeout
20	memout	timeout	timeout	timeout
process with arbiter	UPPAAL breadth first	UPPAAL depth first	FSMT MC interleaving	FSMT MC parallel
5	14.57	50.96	31.07	11.21
6	859.45	timeout	74.18	19.38
7	timeout	timeout	68.95	44.14
8	timeout	timeout	201.44	105.9
9	timeout	timeout	357.68	121.84
10	timeout	timeout	1227.07	175.97
11	timeout	timeout	1708.49	270.04
12	timeout	timeout	timeout	437.15
13	timeout	timeout	timeout	553.91
14	timeout	timeout	timeout	1168.83
15	timeout	timeout	timeout	1177.16
16	timeout	timeout	timeout	2491.45
17	timeout	timeout	timeout	2578.31
18	timeout	timeout	timeout	timeout

TABLE I
EXPERIMENTAL RESULTS

algorithm to verify these FSMTs. In order to verify TAs with our algorithm we presented two different methods to convert TAs into FSMTs. The resulting FSMT has either a pure interleaving behavior or a parallelized interleaving behavior, which can dramatically reduce the number of verification steps for certain benchmark classes and brings an enormous gain of performance. We implemented a prototype of our model checking algorithm and tested it on several benchmarks. The results show that for benchmarks allowing parallelized interleaving behavior our model checking algorithm produces stunning results. On other benchmarks like the well known fischer protocol the variant using pure interleaving is still outperforming the state of the art tool uppaal due to our fully symbolic approach.

REFERENCES

[1] R. Alur. Timed automata. *Theoretical Computer Science*, 126:183–235, 1999.
[2] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.
[3] G. Behrmann, R. David, and K. G. Larsen. A tutorial on uppaal. 2004, pp. 200–236. Springer.
[4] J. Bengtsson and W. Yi. Timed automata: Semantics, algorithms and tools. In W. Reisig and G. Rozenberg, eds., *In Lecture Notes on Concurrency and Petri Nets*, 2004, Lecture Notes in Computer Science vol 3098. Springer–Verlag.
[5] M. Bozzano, R. Bruttomesso, A. Cimatti, T. Junttila, P. van Rossum, S. Schulz, and R. Sebastiani. MathSAT: Tight integration of sat and mathematical decision. *Journal of Automated Reasoning*, 35(1-3):265–293, 2005.
[6] W. Damm, S. Disch, H. Hungar, S. Jacobs, J. Pang, F. Pigorsch, C. Scholl, U. Waldmann, and B. Wirtz. Exact state set representations in the verification of linear hybrid systems with large discrete state space. In *Automated Technology for Verification and Analysis*, Berlin / Heidelberg, 2007, *LNCS 4762*, pp. 425–440. Springer.
[7] W. Damm, S. Disch, H. Hungar, J. Pang, F. Pigorsch, C. Scholl, U. Waldmann, and B. Wirtz. Automatic verification of hybrid systems with large discrete state space. In *Int'l Symp. on Automated Technology for Verification and Analysis*, 2006, *LNCS 4218*, pp. 276–291. Springer.
[8] B. Dutertre and L. de Moura. A fast linear-arithmetic solver for DPLL(T). In *CAV*, 2006, *LNCS 4144*, pp. 81–94. Springer.
[9] K. G. Larsen, P. Pettersson, and W. Yi. Uppaal in a nutshell, 1997.
[10] A. Mishchenko, S. Chatterjee, R. Jiang, and R. K. Brayton. FRAIGs: A unifying representation for logic synthesis and verification. Technical report, EECS Dept., UC Berkeley, 2005.
[11] F. Pigorsch, C. Scholl, and S. Disch. Advanced unbounded model checking by using AIGs, BDD sweeping and quantifier scheduling. In *6th Conference on Formal Methods in Computer Aided Design*, 2006, pp. 89–96. IEEE Press.
[12] C. Scholl, S. Disch, F. Pigorsch, and S. Kupferschmid. Computing optimized representations for non-convex polyhedra by detection and removal of redundant linear constraints. In *Tools and Algorithms for the Construction and Analysis of Systems*, March 2009, *LNCS 5505*, pp. 383–397. Springer.
[13] J. J. Vereijken. Fischer's protocol in timed process algebra, 1994.