Efficient Verification of Timed Automata using BDDs

Dirk Beyer and Andreas Noack

Software Systems Engineering Research Group,
Technical University Cottbus,
D-03013 Cottbus, Postbox 10 13 44, Germany
{db | an}@informatik.tu-cottbus.de

Abstract. This paper investigates the efficient reachability analysis of timed automata. It describes a discretization of time which preserves the reachability properties. The discretization allows to represent sets of configurations of timed automata as binary decision diagrams (BDDs). Further techniques, like computing good variable orderings, are applied to use the full potential of BDDs as compact and canonical representation of large sets. We implemented these concepts within the tool Rabbit. The highly improved performance is shown for some example models. For additional speedup we used an on-the-fly algorithm and refinement checking for large models.

Keywords. Formal verification, Real-time systems, Timed automata, Discretization, BDDs

1 Introduction

Formal specification and verification are commonly used to ensure the correctness of real-time systems. Safety properties can be verified by reachability analysis. Therefore the set of all reachable configurations is computed and the emptiness of its intersection with the set of forbidden configurations is checked.

Timed automata [Alu99] are a modeling formalism for real-time systems. The reachability analysis of timed automata has been implemented in tools like Kronos [BDM$^+$98] and Uppaal [LPY97]. One of the main problems in the application of these tools is the exploding consumption of time for the computation and memory for the representation of the reachable configurations. Thus the data structure for sets of configurations is of vital importance.

Sets of configurations of timed automata consist of locations and associated sets of clock assignments which are subsets of $\mathbb{R}_+^d$. For the symbolic representation of sets of *locations* binary decision diagrams (BDDs) [Bry86] are widely used.

For the representation of sets of *clock assignments* Kronos and Uppaal use difference bound matrices (DBM) [Dil89]. This results in two main performance bottlenecks: DBMs do not provide an efficient representation for non-convex sets, and the use of different data structures for locations and clock assignments often leads to an inefficient representation of configuration sets with many locations.

Asarin et al. introduced the uniform representation of locations and clock assignments as BDDs [ABK$^+$97]. This is based on a discretization of time, i.e. a replacement of the continuous passage of time by a passage in discrete steps, whereby only finitely many representatives of the infinite set of clock assignments are considered [GPV94]. In [AMP98] it was observed that the reachability analysis of so-called *closed* timed automata, whose clock constraints are independent and do not contain the relations $<$ and $>$, needs to consider only integer clock assignments.

Based on this work, we formally define an *integer semantics* of closed timed automata and prove its equivalence to the usual, continuous semantics in Section 3. In Section 4 we explain how the configuration sets and transitions of this integer semantics can be represented as BDDs. In Section 5 we outline some techniques which improve the efficiency of the BDD-based reachability analysis, including BDD variable ordering, on-the-fly analysis and refinement checking. We demonstrate the performance of our tool implementation by applying it to several example models.

2 Timed Automata

This section gives a definition of timed automata and their continuous semantics. We use a formal definition of timed automata similar to that introduced by Alur [Alu99], because it is commonly accepted and provides a good standard.

2.1 Definition

At first, we define **clock constraints**, which are allowed as invariants and guards of a timed automaton. Let $X = \{x_1, \ldots, x_n\}$ be a set of clocks. Atomic clock constraints over X are comparisons of a clock with a time constant from $\mathbb{N}$, the set of natural numbers (including 0). Clock constraints are conjunctions of atomic clock constraints. Formally, the set $\Phi(X)$ of clock constraints over X is generated by the grammar

$$\varphi := x \sim c \mid \varphi \wedge \varphi \mid true,$$

with $x \in X$, $c \in \mathbb{N}$ and $\sim \in \{\leq, \geq, <, >\}$.

A **clock assignment** of X is a total function from X into $\mathbb{R}_+$, where $\mathbb{R}_+$ is the set of non-negative real numbers. $Val(X)$ denotes the set of all clock assignments of X. We define the **semantics of a clock constraint** by the interpretation function $[\![.]\!] : \Phi(X) \to 2^{Val(X)}$, which is inductively defined in the usual way by:

$$\begin{aligned} [\![x \sim c]\!] &:= \{v \in Val(X) \mid v(x) \sim c\} \\ [\![\varphi_1 \wedge \varphi_2]\!] &:= [\![\varphi_1]\!] \cap [\![\varphi_2]\!] \\ [\![true]\!] &:= Val(X), \end{aligned}$$

with $x \in X$, $c \in \mathbb{N}$ and $\sim \in \{\leq, \geq, <, >\}$, i.e. we interpret φ as the set of all clock assignments of X that satisfy φ.

The clock assignment which assigns the value 0 to all clocks is denoted by v^0. For $v \in Val(X)$ and $\delta \in \mathbb{R}_+$, $v + \delta$ is the clock assignment of X that assigns the value $v(x) + \delta$ to each clock x. For $v \in Val(X)$ and $Y \subseteq X$, $v[Y := 0]$ denotes the clock assignment of X that assigns the value 0 to each clock in Y and leaves the other clocks unchanged.

A **timed automaton** $\mathcal{A}$ is a tuple $(L, L^0, X, \Sigma, I, E)$, where

- L is a finite set of **locations**,
- $L^0 \subseteq L$ is a set of **initial locations**,
- X is a finite set of **clocks**,
- Σ, with $\Sigma \cap \mathbb{R}_+ = \emptyset$, is a finite set of **synchronization labels**,
- I is a total function that assigns an **invariant** from $\Phi(X)$ to each location in L,
- $E \subseteq L \times \Sigma \times \Phi(X) \times 2^X \times L$ is a set of **switches**. A switch (l, a, φ, Y, m) represents a transition labeled with synchronization label a from location l to location m. The guard φ has to be satisfied to enable the switch. The switch resets all clocks in Y to the value 0.

Note. There exist different definitions of timed automata. Some of them allow for guards of the form $x \sim y + c$ with $x, y \in X$. Following [Alu99] we do not allow such clock constraints. Thus, we avoid problems with our discretization which requires independent clock constraints.

Complex systems can be described as **parallel composition** of several timed automata which communicate through synchronization labels. A composition of two timed automata with disjoint sets of clocks can be transformed into a single timed automaton by constructing the product automaton. The locations of the product automaton are pairs of component locations, and the invariant of a compound location is the conjunction of the invariants of the component locations. Two switches of the components with the same synchronization label are synchronized.

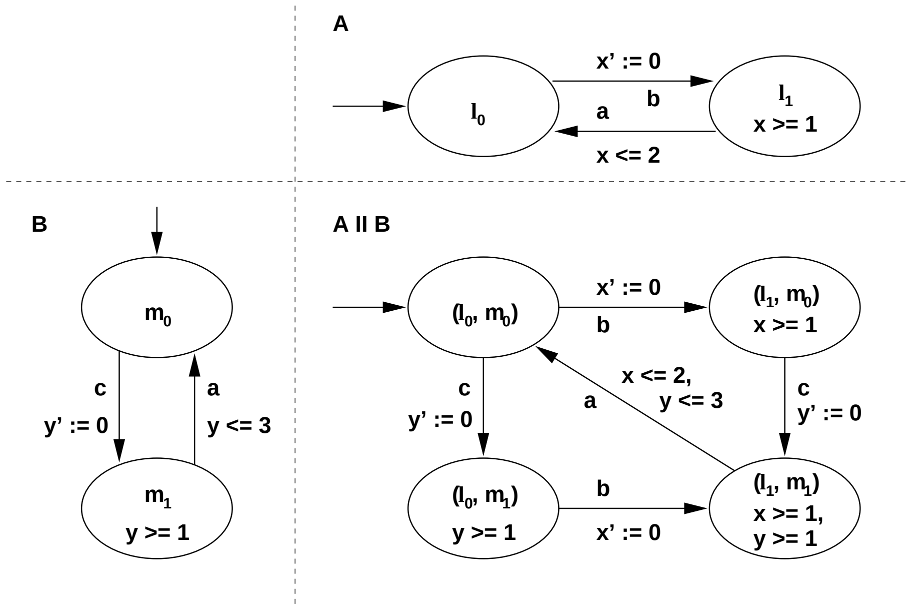

Fig. 1. Two timed automata and their product automaton.

Formally, let $\mathcal{A}_1 = (L_1, L_1^0, X_1, \Sigma_1, I_1, E_1)$ and $\mathcal{A}_2 = (L_2, L_2^0, X_2, \Sigma_2, I_2, E_2)$ be two timed automata, and assume that $X_1 \cap X_2 = \emptyset$. The product automaton $\mathcal{A}_1 \| \mathcal{A}_2$ is the timed automaton $(L_1 \times L_2, L_1^0 \times L_2^0, X_1 \cup X_2, \Sigma_1 \cup \Sigma_2, I, E)$ with $I((l_1, l_2)) = I_1(l_1) \wedge I_2(l_2)$ and $((l_1, l_2), a, \varphi, Y, (m_1, m_2)) \in E$ iff

- $a \in \Sigma_1 \cap \Sigma_2$ and there exist transitions $(l_1, a, \varphi_1, Y_1, m_1) \in E_1$ and $(l_2, a, \varphi_2, Y_2, m_2) \in E_2$, such that $\varphi = \varphi_1 \wedge \varphi_2$ and $Y = Y_1 \cup Y_2$ holds, or
- $a \in \Sigma_1 \setminus \Sigma_2$, $l_2 = m_2$ and there exists a transition $(l_1, a, \varphi, Y, m_1) \in E_1$, or
- $a \in \Sigma_2 \setminus \Sigma_1$, $l_1 = m_1$ and there exists a transition $(l_2, a, \varphi, Y, m_2) \in E_2$.

Figure 1 shows an example for the construction of a product automaton.

2.2 Semantics

The semantics of a timed automaton is defined by associating a transition system with it. A **transition system** $\mathcal{S}$ is a tuple $(Q, Q^0, \Sigma, \rightarrow)$ where Q is the set of configurations, $Q^0 \subseteq Q$ is a set of initial configurations, Σ is a set of labels, and $\rightarrow \subseteq Q \times \Sigma \times Q$ is a set of transitions. The system starts in an initial configuration and can change its configuration from q to q' on label a if $(q, a, q') \in \rightarrow$ (also written as $q \xrightarrow{a} q'$). $q \rightarrow q'$ is written iff $q \xrightarrow{a} q'$ for some label a.

The **continuous semantics** $[\![\mathcal{A}]\!]_C$ of a timed automaton $\mathcal{A} = (L, L^0, X, \Sigma, I, E)$ is the transition system $(L \times \mathit{Val}(X), L^0 \times \{v^0\}, \Sigma \cup \mathbb{R}_+, \rightarrow)$, with $\rightarrow$ containing two kinds of transitions:

- Time transitions:
  For $(l, v), (m, w) \in L \times \mathit{Val}(X)$ and $\delta \in \mathbb{R}_+$, $(l, v) \xrightarrow{\delta} (m, w)$ holds
  iff $l = m$, $w = v + \delta$, $v \in [\![I(l)]\!]$ and $w \in [\![I(l)]\!]$.
- Discrete transitions:
  For $(l, v), (m, w) \in L \times \mathit{Val}(X)$ and $a \in \Sigma$, $(l, v) \xrightarrow{a} (m, w)$ holds
  iff there exists an $(l, a, \varphi, Y, m) \in E$ with $v \in [\![\varphi]\!]$ and $w = v[Y := 0]$.

Note that for all clock constraints $\varphi \in \Phi(X)$ the statements "$v \in [\![\varphi]\!]$ and $v + \delta \in [\![\varphi]\!]$" and "for all $\delta' \in \mathbb{R}$ with $0 \leq \delta' \leq \delta$, $v + \delta' \in [\![\varphi]\!]$ holds" are equivalent. This is true because only conjunctions are allowed as clock constraints.

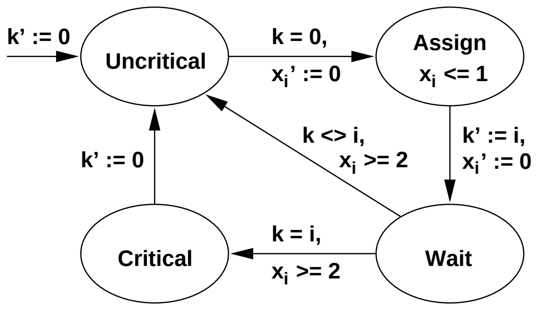

Fig. 2. Process i of Fischer's mutual exclusion protocol.

In the following we define runs and reachable configurations for a transition system $\mathcal{S} = (Q, Q^0, \Sigma_\mathcal{S}, \rightarrow)$. Let $(q_0, q_1, ..., q_k)$ be a finite sequence of configurations, $a_0, a_1, ..., a_{k-1} \in \Sigma_S$, $q_0 \in Q^0$, and $q_i \xrightarrow{a_i} q_{i+1}$ for all $i \in \{0, 1, ..., k-1\}$. Then $(q_0, q_1, ..., q_k)$ is a **run** of $\mathcal{S}$. $Run(\mathcal{S})$ denotes the set of runs of $\mathcal{S}$. The configuration q_k is **reachable**. $Reach(\mathcal{S})$ denotes the set of reachable configurations (shorter reachability set) of $\mathcal{S}$. For $t_1, t_2 \in \mathbb{N}$, the configuration q_k is called reachable between time t_1 and time t_2, if $t_1 \leq \sum_{i \in I} a_i \leq t_2$, with $I = \{j \mid 0 \leq j < k, a_j \in (\Sigma_\mathcal{S} \cap \mathbb{R}_+)\}$ is the set of all indices of time labels from $\{a_0, a_1, ..., a_{k-1}\}$. $Reach(\mathcal{S})(t_1, t_2)$ denotes the set of configurations reachable between time t_1 and t_2.

For a timed automaton $\mathcal{A} = (L, L^0, X, \Sigma, I, E)$ and its continuous semantics $[\![\mathcal{A}]\!]_C = (L \times Val(X), L^0 \times \{v^0\}, \Sigma \cup \mathbb{R}_+, \rightarrow)$ we define the following notions: A location $l \in L$ is **reachable** iff there exists a clock assignment $v \in Val(X)$ with $(l, v) \in Reach([\![\mathcal{A}]\!]_C)$. $ReachLoc([\![\mathcal{A}]\!]_C)$ denotes the set of reachable locations of $\mathcal{A}$ regarding semantics $[\![\mathcal{A}]\!]_C$. Analogous notions we can apply to other semantics of $\mathcal{A}$. Two semantics $[\![\mathcal{A}]\!]_1$ and $[\![\mathcal{A}]\!]_2$ are defined to be **location equivalent** for a timed automaton $\mathcal{A}$ iff $ReachLoc([\![\mathcal{A}]\!]_1) = ReachLoc([\![\mathcal{A}]\!]_2)$.

2.3 Example: Fischer's Protocol

For illustration we explain a model of Fischer's timing-based mutual exclusion protocol [Lam87]. The model is composed from n timed automata like the one depicted in Figure 2, each modeling one process. The processes communicate through a shared variable k with the range $\{0, 1, ..., n\}$. This variable could be modeled as additional automaton, but we prefer a more compact notation. The initial value of k is 0. This value means that no process tries to enter the critical section. When k has a value $i \neq 0$, the process with the identifier i is allowed to enter the critical section or already stays there.

Each process is modeled by an automaton with four locations. Initially it is outside the critical section. If no other process tries to enter the critical section ($k = 0$), it can move to the location Assign. This location models that a process needs at most one time unit to assign its identifier to k. Therefore, the clock x_i measures the staying time in this location, and the invariant forces the automaton to leave the location within one time unit. The transition to the location Wait sets k to the process identifier. After the process has stayed in Wait for two time units, it is guaranteed that no other process is in the location Assign. Now the process is allowed to enter the critical section, if the value of k still is its identifier, otherwise it has to go back to the location Uncritical.

2.4 Reachability Analysis

Figure 3 shows a generic algorithm for reachability analysis, which checks if a given timed automaton $\mathcal{A}$ can reach a location of a given set L^F. During its execution the variable R contains all configurations reached so far. Each iteration adds to R the set $\{q' \in L \times Val(X) \mid \exists q : q \in R \wedge q \rightarrow q'\}$ of configurations reachable from R by taking one transition. The algorithm terminates when:

```
Input: timed automaton $\mathcal{A} = (L, L^0, X, \Sigma, I, E)$
  with the continuous semantics $[\![\mathcal{A}]\!]_C = (L \times Val(X), L^0 \times \{v^0\}, \Sigma \cup \mathbb{R}_+, \rightarrow)$,
  set of locations $L^F$
Output: $true$ iff $L^F \cap ReachLoc([\![\mathcal{A}]\!]_C) \neq \emptyset$
$R := L^0 \times \{v^0\}$
do
  $R_{prev} := R$
  $R := R \cup \{q' \in L \times Val(X) \mid \exists q : q \in R \wedge q \rightarrow q'\}$
  if $R \cap (L^F \times Val(X)) \neq \emptyset$ then return $true$
while $R \neq R_{prev}$
return $false$
```

Fig. 3. Algorithm for reachability analysis.

1. A configuration is reached whose location is one of the forbidden locations in L^F. Then the algorithm stops and notifies the violation of the safety property.
2. The fixed point $R = R_{prev}$ is reached. Then R contains all reachable configurations. If there is no location component from L^F (case one) then the safety property is fulfilled.

The main goal of the following sections is to explain an efficient BDD-based implementation of this reachability algorithm. The algorithm remains correct if it uses a semantics which is *location equivalent* to the continuous semantics. The next section proves this location equivalence for an integer semantics with a finite number of configurations. Section 4 describes how the configuration sets and transitions of this integer semantics can be represented as BDDs.

3 Integer Semantics

A discretization of time which is location equivalent to the continuous semantics exists for all timed automata [GPV94]. However, we restrict ourselves to the subclass of closed timed automata to allow for a discretization which is particularly simple and enables efficient reachability analysis. (For DBMs and similar data structures, this does not lead to significant performance improvements.) This restriction is of technical nature, and we did not found examples within our application area of production cells and real-time algorithms for which it is difficult to construct models using only non-strict constraints with integer constants.

Closed timed automata have only clock constraints φ generated by $\varphi := x \leq c \mid x \geq c \mid \varphi \wedge \varphi$ with $x \in X$ and $c \in \mathbb{N}$, i.e. the relations $<$ and $>$ are not allowed. The product automaton of two closed timed automata is closed again. For closed timed automata it is sufficient to use only integer clock values for the computation of reachable locations. For a set of clocks X the set of integer clock assignments $Val_I(X)$ is defined to be the set of total functions from X to $\mathbb{N}$. For a timed automaton $\mathcal{A}$ with a clock x, $C_{\mathcal{A}}(x)$ denotes the greatest constant x is compared with in a clock constraint of $\mathcal{A}$. For $v \in Val_I(X)$ and $\delta \in \mathbb{N}$, $v \oplus \delta$ is the clock assignment of X that assigns the value $min\,(v(x) + \delta, C_{\mathcal{A}}(x) + 1)$ to each clock x. The definition of the integer semantics is analogous to the continuous semantics.

Let $\mathcal{A} = (L, L^0, X, \Sigma, I, E)$ be a closed timed automaton. The **integer semantics** $[\![\mathcal{A}]\!]_I$ of $\mathcal{A}$ is the transition system $(L \times Val_I(X), L^0 \times \{v^0\}, \Sigma \cup \mathbb{N}, \rightarrow_I)$ with the following transitions:

– Time transitions:
  For $(l, v), (m, w) \in L \times Val_I(X)$ and $\delta \in \mathbb{N}$, $(l, v) \xrightarrow{\delta}_I (m, w)$ holds iff $l = m$, $w = v \oplus \delta$, $v \in [\![I(l)]\!]$ and $w \in [\![I(l)]\!]$.
– Discrete transitions:
  For $(l, v), (m, w) \in L \times Val_I(X)$ and $a \in \Sigma$, $(l, v) \xrightarrow{a}_I (m, w)$ holds iff there exists an $(l, a, \varphi, Y, m) \in E$ with $v \in [\![\varphi]\!]$ and $w = v[Y := 0]$.

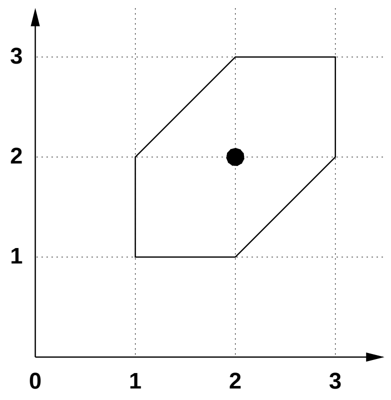

Fig. 4. The set of continuous clock assignments represented by the integer clock assignment (2, 2).

To prove the *location equivalence* of the integer and the continuous semantics, we define for a timed automaton $\mathcal{A}$ with the set of clocks X the relation $\succ \subseteq Val(X) \times Val_I(X)$ associating every continuous clock assignment with its possible integer **representatives**. For $v \in Val(X)$ and $v' \in Val_I(X)$, $v \succ v'$ holds iff there exists some $\gamma \in \mathbb{R}$ with $0 \leq \gamma < 1$, such that for each clock $x \in X$ the following holds:

a) $v'(x) - 1 + \gamma < v(x) \leq v'(x) + \gamma$, or
b) $v'(x) - 1 + \gamma < v(x)$ and $v'(x) = C_{\mathcal{A}}(x) + 1$.

Thus, v' is a representative of v if v' results from v by rounding off all clock values with fractional parts smaller than or equal to a certain bound and by rounding up all clock values with fractional parts greater than this bound in the first case. The second case restricts the range of the representatives to $C_{\mathcal{A}}(x) + 1$. This is sufficient to ensure the crucial property of the representative relation: If a continuous clock assignment v satisfies a clock constraint from $\mathcal{A}$, then its integer representatives satisfy this constraint, too. To illustrate the representative relation, Fig. 4 displays the set of real clock assignments for which the integer point $(2, 2)$ is a representative.

Proofs of the *location equivalence* of the integer semantics and the continuous semantics for other formalisms than timed automata can be found in [HMP92] and [AMP98]. Our proof for timed automata starts with the following lemma.

Lemma 1. *Let* $\mathcal{A} = (L, L^0, X, \Sigma, I, E)$ *be a closed timed automaton with the continuous semantics* $[\![\mathcal{A}]\!]_C = (L \times Val(X), L^0 \times \{v^0\}, \Sigma \cup \mathbb{R}_+, \rightarrow_C)$ *and the integer semantics* $[\![\mathcal{A}]\!]_I = (L \times Val_I(X), L^0 \times \{v^0\}, \Sigma \cup \mathbb{N}, \rightarrow_I)$. *Then the following holds:*

1. *Let* $(l, v), (l, w) \in L \times Val(X), \delta \in \mathbb{R}_+$, *such that* $(l, v) \xrightarrow{\delta}_C (l, w)$ *holds. Then for all* $v' \in Val_I(X)$ *with* $v \succ v'$ *there exists a* $\delta' \in \mathbb{N}$ *and a* $w' \in Val_I(X)$, *such that* $(l, v') \xrightarrow{\delta'}_I (l, w')$ *and* $w \succ w'$ *holds.*
2. *Let* $(l, v), (m, w) \in L \times Val(X), a \in \Sigma$, *such that* $(l, v) \xrightarrow{a}_C (m, w)$ *holds. Then for all* $v' \in Val_I(X)$ *with* $v \succ v'$ *there exists a* $w' \in Val_I(X)$ *such that* $(l, v') \xrightarrow{a}_I (m, w')$ *and* $w \succ w'$ *holds.*

Before we give a formal proof of the lemma we describe the idea of the first statement informally. Therefore we consider the example in Figure 5 which shows a transition of time $\delta = 0.8$ from $(0.8, 1.3)$ to $(1.6, 2.1)$. The integer point $(1, 1)$ is a representative of the starting point $(0.8, 1.3)$ for e.g. $\gamma = 0.6$.

Moving from $(0.8, 1.3)$ to $(1.6, 2.1)$, the point $(1.5, 2.0)$ is the first which is not represented by $(1, 1)$. But the points from immediately after $(1, 1.5)$ to the end point $(1.6, 2.1)$ are represented by $(2, 2)$. Thus, choosing $\delta' = 1$ we get a transition from $(1, 1)$ (which is our example of the representatives of the starting point $(0.8, 1.3)$) to a representative of the end point $(1.6, 2.1)$.

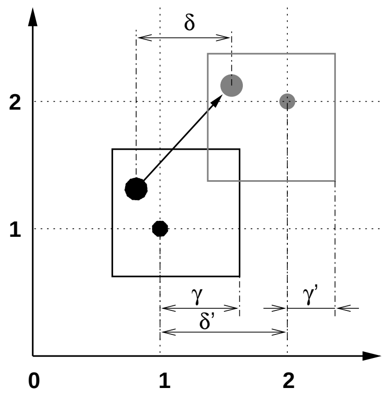

Fig. 5. Discretization and representatives.

Generally, the end point of the δ'-transition is a representative of the end point of the δ-transition if we choose $\delta' = \lfloor \delta + \gamma \rfloor$.

Proof. Statement 1: We have to distinguish the two cases of the definition of $\succ$. Let $v' \in Val_I(X)$, $v \succ v'$. Then according to the definition of the relation $\succ$ there exists some $\gamma \in \mathbb{R}$ with $0 \leq \gamma < 1$, such that the following holds for all $x \in X$:

Case a) $v'(x) + \gamma + \delta < C_{\mathcal{A}}(x) + 1$:

$$v'(x) - 1 + \gamma < v(x) \ \leq v'(x) + \gamma.$$

Because $w = v + \delta$, the following holds for all $x \in X$:

$$v'(x) - 1 + \gamma + \delta < w(x) \ \leq v'(x) + \gamma + \delta.$$

Let $\delta' = \lfloor \delta + \gamma \rfloor$ and $w' = v' + \delta'$. Then for all $x \in X$ the following holds:

$$w'(x) - \delta' - 1 + \gamma + \delta < w(x) \ \leq w'(x) - \delta' + \gamma + \delta.$$

Because $0 \leq \gamma + \delta - \delta' < 1$, this implies $w \succ w'$.

Case b) $v'(x) + \gamma + \delta \geq C_{\mathcal{A}}(x) + 1$:
Using $\delta' = \lfloor \delta + \gamma \rfloor$ and $w' = v' \oplus \delta'$, we obtain analogously to Case a)

$$w'(x) - \delta' - 1 + \gamma + \delta < w(x),$$

and thus, $w \succ w'$.

Because v and w satisfy the invariant $I(l)$ and $v \succ v'$ and $w \succ w'$ hold, we can conclude that v' and w' satisfy the invariant $I(l)$. Thus, we get $(l, v') \xrightarrow{\delta'}_I (l, w')$.

Statement 2: Because of the definition of the continuous semantics, there exists a $(l, a, \varphi, Y, m) \in E$ with $v \in [\![\varphi]\!]$ and $w = v[Y := 0]$. Let $v' \in Val_I(X)$, $v \succ v'$. Then $v' \in [\![\varphi]\!]$ because $v \in [\![\varphi]\!]$. Observe that $v \succ v'$ implies $v[Y := 0] \succ v'[Y := 0]$. Thus, setting w' to $v'[Y := 0]$ we get statement 2. □

Theorem 1. *For every closed timed automaton $\mathcal{A}$, $ReachLoc([\![\mathcal{A}]\!]_C) = ReachLoc([\![\mathcal{A}]\!]_I)$ holds.*

Proof. Let $\mathcal{A} = (L, L^0, X, \Sigma, I, E)$ be a timed automaton with the continuous semantics $[\![\mathcal{A}]\!]_C = (L \times Val(X), L^0 \times \{v^0\}, \Sigma \cup \mathbb{R}_+, \rightarrow_C)$ and the integer semantics $[\![\mathcal{A}]\!]_I = (L \times Val_I(X), L^0 \times \{v^0\}, \Sigma \cup \mathbb{N}, \rightarrow_I)$.

At first, we prove $ReachLoc([\![\mathcal{A}]\!]_C) \subseteq ReachLoc([\![\mathcal{A}]\!]_I)$. We show per induction over k that for every run $((l_0, v_0), (l_1, v_1), ..., (l_k, v_k))$ in $Run([\![\mathcal{A}]\!]_C)$ there exists a run $((l_0, v'_0), (l_1, v'_1), ..., (l_k, v'_k))$ in $Run([\![\mathcal{A}]\!]_I)$, such that $v_i \succ v'_i$ holds for all $i \in \{0, 1, ..., k\}$.

Start of induction: According to the definition of run, $l_0 \in L^0$ and $v_0 = v^0$ hold. $((l_0, v^0))$ is also in $Run([\![\mathcal{A}]\!]_I)$, and $v_0 \succ v^0$ holds.

Inductive step: We have to show that there exists some $a' \in \Sigma \cup \mathbb{N}$ and some v'_{i+1} with $v_{i+1} \succ v'_{i+1}$, such that $(l_i, v'_i) \xrightarrow{a'}_I (l_{i+1}, v'_{i+1})$. The inductive hypothesis ensures $v_i \succ v'_i$, and

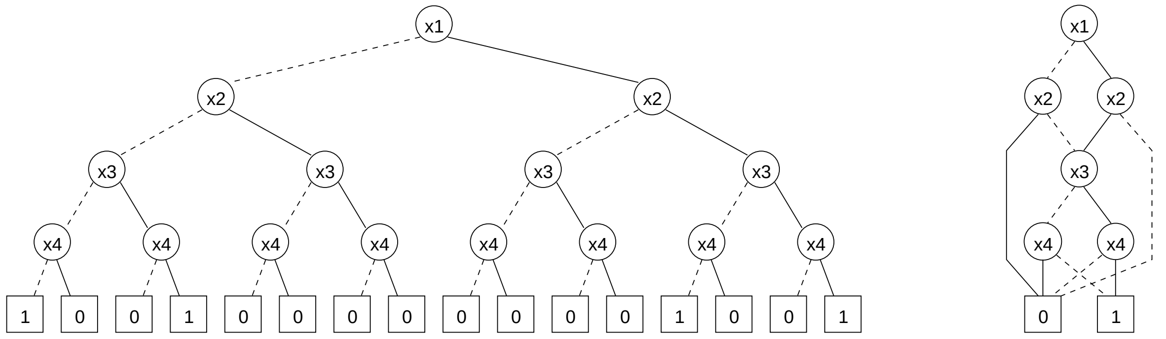

Fig. 6. Decision tree and BDD of a set of assignments.

there exists some $a \in \Sigma \cup \mathbb{R}_+$ with $(l_i, v_i) \xrightarrow{a}_R (l_{i+1}, v_{i+1})$. The assertion of the theorem follows from Lemma 1, statement 1, if $a \in \mathbb{R}_+$, and statement 2, if $a \in \Sigma$. This finishes the inductive proof.

$ReachLoc(\llbracket\mathcal{A}\rrbracket_I) \subseteq ReachLoc(\llbracket\mathcal{A}\rrbracket_C)$ follows directly from the definitions of the semantics.

□

4 BDD-based Reachability Analysis

The first subsection informally introduces binary decision diagrams. It describes how sets of assignments of Boolean variables can be represented as BDDs. The second subsection introduces some notations for discrete variables which are a generalization of Boolean variables. The last subsection shows that sets of configurations and transitions of the integer semantics can be regarded as assignment sets of discrete variables, and how the BDD representations of these assignment sets can be computed.

4.1 Binary Decision Diagrams

A binary decision diagram (BDD) [Bry86] represents a set of assignments for a set of Boolean variables. BDDs give canonical and compact representations of sets and allow an efficient implementation of operations like intersection, union and existential quantification.

A BDD is a directed acyclic graph which is derived by reducing a binary decision tree. A binary decision tree consists of decision nodes, 0-terminal-nodes and 1-terminal-nodes. Each decision node is assigned to a Boolean variable and has two children called low child and high child.

The assignments represented by the decision tree correspond to the paths from the root node to the 1-terminal-nodes. The variable of a node has the value 0 if the path descends to the low child and the value 1 if the path descends to the high child.

The bottom-up application of the following two reduction rules transforms a binary decision tree into a BDD:

1. Merge any isomorphic subtrees.
2. Eliminate any node whose two children are isomorphic.

In this paper we only deal with ordered BDDs (also called OBDDs) which means that the variables occur in the same order on any path from the root to a terminal node. For a given variable ordering, the representation of a set of assignments is unique.

Figure 6 shows the decision tree and BDD representation of the set of all assignments v of the Boolean variables $\{x_1, x_2, x_3, x_4\}$ with $v(x_1) = v(x_2)$ and $v(x_3) = v(x_4)$.

4.2 Discrete Variables

Let $Range(x)$ denote the range of a variable x. **Discrete variables** are variables with a finite and nonempty range. Boolean variables are discrete variables with the range $\{0, 1\}$.

Obviously, a BDD can not only represent an assignment set of Boolean variables, but also an assignment set of discrete variables. Therefore every discrete variable x with $|Range(x)| > 2$ is encoded by several Boolean variables.

Let X be a set of discrete variables. The set $\Phi(X)$ of **constraints** over X is generated by the grammar $\varphi := x \sim c \mid \varphi \wedge \varphi$, with $x \in X$, $\sim \in \{\leq, \geq, <, >, =\}$ and $c \in Range(x)$.

An **assignment** of X is a function which assigns an element of $Range(x)$ to each variable $x \in X$. $Val(X)$ denotes the set of all assignments of X. For a constraint $\varphi \in \Phi(X)$, $[\![\varphi]\!]$ denotes the set of all assignments of X that satisfy φ. This definition is ambiguous because for two sets of clocks X and Y, $\varphi \in \Phi(X)$ implies $\varphi \in \Phi(X \cup Y)$, and thus $[\![\varphi]\!]$ could be an element of $Val(X)$ as well as of $Val(X \cup Y)$. We resolve this ambiguity by identifying the corresponding assignment sets. Formally: $V \subseteq Val(X)$ and $W \subseteq Val(X \cup Y)$ are identified iff $W = \{w \in Val(X \cup Y) \mid \exists v \in V. \forall x \in X. v(x) = w(x)\}$. This is justified by the fact that V and W have the same BDD representation.

Now we introduce two notations for a set of assignments $V \subseteq Val(X)$. For a variable $x \in X$, the **existential quantification** $\exists x.V$ is defined to be the set of all assignments of $X \setminus \{x\}$ which agree in the values of all variables but x with an assignment in V. Formally: For $w \in Val(X \setminus \{x\})$, $w \in \exists x.V$ holds iff there exists some $v \in V$, such that $v(y) = w(y)$ for all $y \in X \setminus \{x\}$.

For two variables $x \in X$ and $y \notin X$, $V[x \leftarrow y]$ is the set of assignments which is obtained by **renaming** x by y. Formally: For $w \in Val((X \setminus \{x\}) \cup \{y\})$, $w \in V[x \leftarrow y]$ holds iff there exists some $v \in V$, such that $v(x) = w(y)$ and $v(z) = w(z)$ for all $z \in X \setminus \{x\}$.

There exist efficient BDD operations for existential quantification and renaming of variables, as well as for intersection, union and comparison of sets of assignments.

4.3 Representing Configurations and Transitions as BDDs

For a BDD-based implementation of the reachability algorithm in Figure 3, the transition relation has to be represented by BDDs. A finite relation can be transformed into a set of assignments by mapping the arguments of the relation to discrete variables. Let $P \subseteq P_1 \times P_2 \times ... \times P_n$ be a relation and $X = \{x_1, x_2, ..., x_n\}$ a set of discrete variables with $Range(x_i) = P_i$ for all $i \in \{1, 2, ..., n\}$. Then $V_P(x_1, x_2, ..., x_n)$ denotes the set of assignments of X with $v \in V_P(x_1, x_2, ..., x_n)$ iff $(v(x_1), v(x_2), ..., v(x_n)) \in P$.

The core operation of the reachability analysis is, given a transition system $(Q, Q^0, \Sigma, \rightarrow)$ and a set of configurations $R \subseteq Q$, to compute the set of successor configurations $R' = \{q' \in Q \mid \exists q : q \in R \wedge q \rightarrow q'\}$. Using two variables q, q' with $Range(q) = Range(q') = Q$, the set of configurations R can be transformed into the set of assignments $V_R(q)$ with domain $\{q\}$, and $\rightarrow$ into the set of assignments $V_\rightarrow(q, q')$ of $\{q, q'\}$. Then $V_R(q) \cap V_\rightarrow(q, q')$ is a set of assignments which associates each element of R with its successor configurations: $v \in V_R(q) \cap V_\rightarrow(q, q')$ iff $v(q) \in R$ and $v(q) \rightarrow v(q')$. The existential quantification of q results in the desired set of successor configurations as set of assignments of q': $v \in \exists q.(V_R(q) \cap V_\rightarrow(q, q'))$ iff $v(q') \in R'$, or in other words, $V_{R'}(q') = \exists q.(V_R(q) \cap V_\rightarrow(q, q'))$. Now we can transform this set into a set of assignments of q by renaming q' to q: $V_{R'}(q) = (\exists q.(V_R(q) \cap V_\rightarrow(q, q')))[q' \leftarrow q]$. Shortly, we write $\rightarrow(q, q')$ for $V_\rightarrow(q, q')$ and $\xrightarrow{a}(q, q')$ for $V_{\xrightarrow{a}}(q, q'), a \in \Sigma$ in the sequel of the paper.

The remaining problem is to compute the BDD representation of the transition relation for a given closed timed automaton $\mathcal{A} = (L, L^0, \{x_1, ..., x_n\}, \Sigma, I, E)$ with the integer semantics $[\![\mathcal{A}]\!]_I = (Q, Q^0, \Sigma \cup \mathbb{N}, \rightarrow)$.

The transition relation is represented by several BDDs: one BDD for the time transition relation $\xrightarrow{1}$ and one BDD for the discrete transition relation $\xrightarrow{a}$ of each synchronization label

$a \in \Sigma$. These partial transition relations can be represented as assignment sets of the variables $\{l, l', x_1, x_1', ..., x_n, x_n'\}$. Here a configuration of $\mathcal{A}$ is represented by the location and the integer clock values. The variable l with $Range(l) = L$ contains the location, and the clocks x_i ($1 \leq i \leq n$) are regarded as discrete variables with $Range(x_i) = \{0, 1, ..., C_{\mathcal{A}}(x_i) + 1\}$. The primed version of each variable contains its value in the successor configuration and has the same range as the unprimed version. Using these notations, the representations of the transition relations can be computed as follows:

$$\xrightarrow{1}(l, l', x_1, x_1', ..., x_n, x_n') = [\![l' = l]\!] \cap [\![I(l)]\!] \cap [\![I(l)]\!][x_1 \leftarrow x_1', ..., x_n \leftarrow x_n'] \cap \bigcap_{i \in \{1,2,...,n\}} [\![x_i' = min(x_i + 1, C_{\mathcal{A}}(x_i) + 1)]\!]$$

and for all $a \in \Sigma$

$$\xrightarrow{a}(l, l', x_1, x_1', ..., x_n, x_n') = \bigcup_{(m,a,\varphi,Y,m') \in E} \left([\![l = m]\!] \cap [\![l' = m']\!] \cap [\![\varphi]\!] \cap \bigcap_{i \in \{1,2,...,n\}} \begin{cases} [\![x_i' = 0]\!], & \text{if } x_i \in Y \\ [\![x_i' = x_i]\!], & \text{otherwise} \end{cases} \right)$$

To compute the transition relation for the product automaton $\mathcal{A}$ of several timed automata $\mathcal{A}_1, \mathcal{A}_2, ..., \mathcal{A}_n$ with pairwise disjoint sets of clocks it is not necessary to compute the product automaton explicitly. Let Σ_i be the set of synchronization labels and $\rightarrow_i$ the transition relation of $\mathcal{A}_i$ ($i \in \{1, 2, ..., n\}$). A discrete transition of $\mathcal{A}$ with synchronization label a means that all components having a in their alphabet do an a-transition, while the other components do not change their configuration. Let q_i and q_i' be the variables for the configuration and successor configuration of $\mathcal{A}_i$, respectively. Then the following holds for each synchronization label a in the set of synchronization labels $\bigcup_{i \in \{1,2,...,n\}} \Sigma_i$ of the product automaton:

$$\xrightarrow{a}(q_1, q_1', ..., q_n, q_n') = \bigcap_{i \in \{1,2,...,n\}} \begin{cases} \xrightarrow{a}_i(q_i, q_i'), & \text{if } a \in \Sigma_i \\ [\![q_i' = q_i]\!], & \text{otherwise} \end{cases}$$

For the time transition relation of $\mathcal{A}$ the following holds:

$$\xrightarrow{1}(q_1, q_1', ..., q_n, q_n') = \bigcap_{i \in \{1,2,...,n\}} \xrightarrow{1}_i(q_i, q_i').$$

Time transitions of more than one time unit can be represented as a sequence of transitions of one time unit. One monolithic transition relation could be obtained by uniting the partial transition relations, but in practice it is more efficient to apply the partial relations sequentially (cf. Section 5).

5 Efficient Implementation

Based on the concepts described in the last two sections we extended our existing matrix-based model checker **Rabbit** [BR00] by the BDD-based reachability analysis of closed timed automata. Experience with finite automata shows that the efficiency critically depends on the choice of several parameters [RAB$^+$95]. In this section we sketch how our implementation determines these parameters.

As stated in Subsection 4.1, the BDD representation of a configuration set is unique for a fixed variable ordering. However, changing the **variable ordering** can have a big impact on the BDD's size. We outline our approach for finding good variable orderings in Section 5.1.

In Subsection 4.3 we suggested to represent the transition relation $\rightarrow$ as implicit union of a timed transition relation $\xrightarrow{1}$ and discrete transition relations $\xrightarrow{a}$ for each synchronization label a.

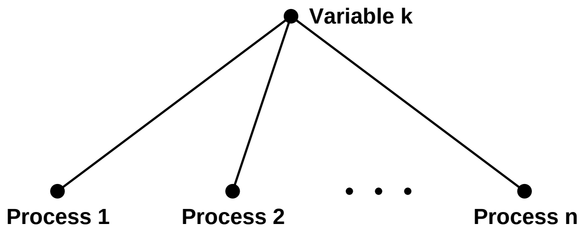

Fig. 7. Communication graph of Fischer's protocol.

Experiments have shown that applying such **partial transition relations** sequentially is more efficient than using the union of these relations as monolithic transition relation.

Using several partial transition relations, we have to determine the **order of** their **application**. The intermediate sets of reached configurations in the reachability algorithm (intermediate values of R in Figure 3) depend on this ordering, and therefore the size of the intermediate BDDs. A bad ordering of the partial transition relations can result in intermediate BDDs that are much bigger than the final BDD of all reachable configurations. Always computing the fixed point using only discrete transitions before applying time transitions is a successful strategy to avoid this problem.

Additional performance improvements can be achieved by using an **On-the-fly algorithm** which checks all reachable configurations without storing the whole set (cf. Section 5.2). For modular verification we provide a **refinement check** based on the existence of a simulation relation.

5.1 Variable Ordering

Because finding optimal variable orderings is algorithmically intractable [BW96], we need to apply heuristics. Analogously to Aziz' approach for communicating finite automata [ATB94], our heuristics is based on a size estimate for the BDD of the set of reachable configurations. This estimate is derived from our upper bound for the transition relation as described and proven in [Bey01b]. We apply the arbitrary insertion heuristic [LLKS85] to optimize the variable ordering with respect to the size estimate.

The size estimate is based on the communication structure of the model. Two automata are communicating if the intersection of their sets of synchronization labels is not empty. Considering the automata as nodes and communication between automata as edges we obtain the communication graph. Fig. 7 illustrates the communication graph of Fischer's protocol for n processes. The size estimate that we discuss in the following contains two general characteristics of good variable orderings, which we can evaluate using the communication graph: **(1)** Communicating components have successive positions within the ordering. **(2)** Components which communicate with many other components precede these other components within the ordering. This leads to the conclusion that the BDD-based representation is quite efficient for models with a weakly connected communication graph like trees and rings. Such models do not have to be regular-structured. For the most models, especially for our production cell, that we have analyzed this is the case.

Let $(\mathcal{A}_1, \mathcal{A}_2, ..., \mathcal{A}_n)$ be an ordered set of components and let $Comm_{\mathcal{A}}(i)$ be the set containing the indices of all components $\mathcal{A}_k$ of the product automaton $\mathcal{A}$ which have an index less than i and communicate with a component having an index greater than or equal to i: $Comm_{\mathcal{A}}(i) = \{k \mid k < i$ and there exists an $l \geq i$ with $\mathcal{A}_k$ communicates with $\mathcal{A}_l\}$. We define the size estimate for the BDD representing the reachable set as follows ($|q_i|$ is the number of bits needed to represent a configuration of $\mathcal{A}_i$, $|Q_k|$ is the number of configurations within the state space of component $\mathcal{A}_k$):

$$\sum_{i=1}^{n} \left(2^{|q_i|} - 1\right) \cdot \left(4 \cdot \prod_{k \in Comm_{\mathcal{A}}(i)} |Q_k| + 4\right).$$

No. of processes	4	5	6	7	8	10	12	14	16	32	64	128
Clocks separated	0.3	1.0	5.0	21.6	110							
BDD size actual	828	3053	10983	38515	132245							
BDD size estimate	4518	21157	96932	436900	1944230							
Variable k at end	0.3	0.6	1.6	3.9	9.4	46.3	249					
BDD size actual	456	1003	2119	4625	9158	36405	145438					
BDD size estimate	2215	8871	35495	158375	633511	10136200	162180000					
Variable k in front	0.3	0.4	0.8	1.3	2.3	4.0	8.9	13.6	22.7	208	1920	9168
BDD size actual	326	544	812	1129	1497	2375	3450	4720	6190	24983	100200	401161
BDD size estimate	561	869	1243	1684	2190	3400	4874	6612	8615	34136	135272	591600

Table 1. Computation times for the verification of the mutex property of **Fischer's protocol**. Computation time of the experiments using Rabbit is given in seconds of CPU time on a SUN Ultra-Sparc 1 with 200 MHz processor and 512 MB memory in all tables.

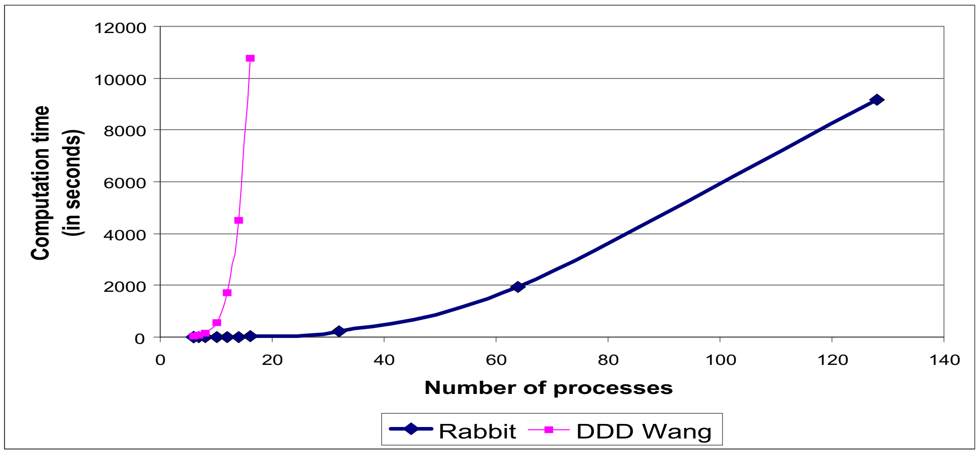

Fig. 8. Computation times depending on the number of **Fischer processes**. Computation time of Wang's tool is taken from [Wan00] (Pentium II, 366 MHz).

Fischer's protocol: Some experiments with Fischer's protocol (cf. Sec. 2.3) validate the soundness of this estimate: Changing the positions of the process automata in the variable ordering has no effect on the estimate of the BDD's size; only the position of the variable k is important and that the encodings of clock and location of an automaton have neighboring positions (cf. Fig. 7). The first three entries of Table 1 report the results of our experiments with different variable orderings. We examined three strategies. The first row of an experiment in the table contains the computation time in seconds, the second row indicates the growth of the actual size of the BDD representing all reachable configurations and the third row contains the computed value of our size estimate for that BDD.

In the first experiment we used a variable ordering violating the heuristic rule that the variables of a component have successive positions. We used the variable ordering (variable k, automaton 1, ..., automaton n, clock 1, ..., clock n). It leads to a very fast growth of the BDD's size.

If we place the variable k on the last position we get $Comm_1 = Comm_{n+2} = \emptyset$ and $Comm_i = \{1, ..., i-1\}$ for $i \in \{2, ..., n+1\}$. We can compute the estimate for the BDD $\mathcal{B}$ of all reachable configurations as follows: We start to compute $|\mathcal{B}|_i := \prod_{k \in Comm_{\mathcal{A}}(i)} |Q_k|$ (we can leave out the 4 because of using the O-notation). Since $Comm_1 = Comm_{n+2} = \emptyset$, we get the estimate 1 for $|\mathcal{B}|_1$ and $|\mathcal{B}|_{n+2}$. For $|\mathcal{B}|_i$ ($i \in \{2, ..., n+1\}$) we get 12^{i-1} since $|Q_k| = 12$ for $k \in \{1, ..., n\}$ (number of configurations for each process = four locations $\times$ three clock values[1]) and $Comm_i = \{1, ..., i-1\}$

[1] We used three clock values within the model, and the tool implementation uses the range given by the range definition within the model (instead of $C_{\mathcal{A}}(x) + 1$, which would be 4). Thus, the measurements of Table 1 are based on three clock values. Considering the definition of $v(x) \oplus \delta$ we would have to use four clock values, which leads to $|Q_k| = 16$ instead of $|Q_k| = 12$.

Number of input signals	2	4	8	16
Rabbit's computation time	0.5	6.0	79.6	1208.7

Table 2. Time for computation of all reachable configurations of the **AND** model.

for $i \in \{2, ..., n+1\}$. Because variable k is on position $n+1$, the biggest term of the sum for the estimate is that for *component k*, which is $\left(2^{|q_{n+1}|} - 1\right) \cdot 12^n$. The estimate for $|\mathcal{B}|$ therefore is in $O(n \cdot 12^n)$. The second experiment shows that the BDD's size actually grows exponentially.

Placing variable k on first position we have $Comm_1 = Comm_{n+2} = \emptyset$, $Comm_2 = ... = Comm_{n+1} = \{1\}$. The estimate for $|\mathcal{B}|_1$ and $|\mathcal{B}|_{n+2}$ is 1 again, but the estimate for $|\mathcal{B}|_2, ..., |\mathcal{B}|_{n+1}$ is $n+1$ (because $|Q_1| = n+1$). Thus, the estimate for the size of $\mathcal{B}$ is in $O(n^2)$. In the third experiment the estimate matches the actual size (number of nodes) very good. The computation time needed by our tool is polynomial for this example (cf. Fig. 8).

To understand the relative large difference between the estimate and the actual size of the BDD of the first two cases, consider the following: Let $\mathcal{B}$ be the BDD over $(q_1, ..., q_n)$ which represents $Reach(\llbracket \mathcal{A} \rrbracket_I)$. For $i \in \{1, ..., n\}$, let the variable q_i be encoded by the Boolean variables $x_{i,1}, ..., x_{i,|q_i|}$, such that $\mathcal{B}$ is a BDD over $(x_{1,1}, ..., x_{1,|q_1|}, ..., x_{n,1}, ..., x_{n,|q_n|})$. The estimate contains the pessimistic bound that the number of $x_{i,k+1}$ nodes in $\mathcal{B}$ is twice the number of $x_{i,k}$ nodes ($1 \leq k < |q_i|$). This assumption is not realistic for variables q_i with large number of bits and the estimate is not very close to the actual size. In the third experiment q_i does not blow up so intensively, because after such a wideness within the BDD follows the q_{i+1} for a connected component, which make the BDD thin (i.e. it "consumes" the state space of the previous component). To get a better estimate for the number of $x_{i,k+1}$ nodes we can also use a linear or exponential interpolation. But for our purpose the estimate is sufficient because it reflects the relation between different variable orderings appropriately.

It would be unfair to compare these results with matrix-based tools, e.g. Kronos, Uppaal and HyTech, because they are at least exponential in the number of processes as long as they use explicit enumeration for the locations of the automata. An advanced version of Uppaal that is based on compositional and symbolic model checking techniques is able to verify up to 9 Fischer processes (cf. [LPY95], p. 11).

The BDD-based version of Kronos is able to verify 14 processes. All the performance results of Kronos are reported in [BMPY97] and are also obtained using a SUN Ultra-Sparc-1. Wang implemented a tool using another BDD-like data structure called DDD. The computation time of this tool is also depicted in Fig. 8 (cf. [Wan00], p. 12).

One of the most important advantage of our strategy is that we can merge the discrete state spaces with the continuous one. This enables the usage of all possible variable orderings. In difference to the most existing BDD applications we prefer to use static ordering based on the estimate instead of dynamic reordering, which consumes a lot of run-time.

AND circuit: Table 2 shows the results applying our tool to compute the whole set of reachable configurations of the AND model as mentioned in [BMPY97]. This model has a more complicated communication graph than Fischer's protocol. The BDD-based version of Kronos needs 324.7 seconds for the AND model with four inputs.

Production cell: To validate the practical relevance of our tool using a complex system, we developed a CTA model of a production cell, which is similar to the Lewerentz/Lindner production cell from FZI [LL95]. This system consists of 20 machines and belts with 44 sensors and 28 motors. We modeled the system as modular composition of several belts, turntables and machines, including 45 timed automata with 22 clocks.

Fig. 9 shows two BDDs for the reachable set using the production cell model. We visualize a BDD in the following way: each pixel represents one BDD node and each horizontal line represents all the nodes for one bit of a variable (one level within the BDD graph), that means a long

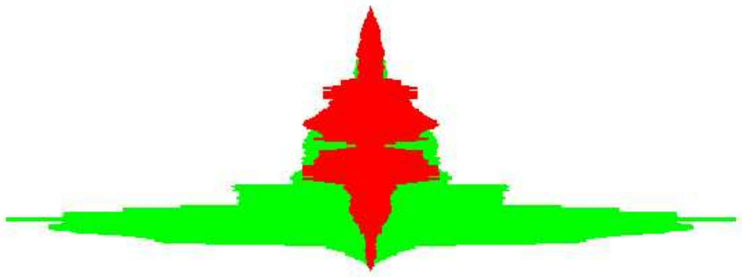

Fig. 9. The BDD shape for the full reachable set for two different variable orderings for the **production cell** model.

horizontal line represents a BDD level with a lot of BDD nodes. In the fi rst experiment we used the variable ordering regarding the modular structure of the model, but the estimation-based heuristic is not applied. Therefore the knowledge of the modeler about the coupling between components is respected and the resulting variable ordering is better than a random ordering. It demonstrates how the BDD grows very fast within the last components (large shape). If we apply our heuristic, we can get a better variable ordering (small shape). The computation of that variable ordering is done automatically by our tool.

5.2 On-the-fly Analysis

This subsection describes a useful technique to avoid the exploding BDD sizes which results from timing-based dependencies. Even if the components do not communicate directly, the BDD representation of the reachable states can be very large. The reason is that their confi gurations depend on the passage of time and therefore, the components are strongly connected in this indirect way.

In contrast to the standard algorithm, which computes at fi rst the set of all reachable confi gurations and then checks whether the intersection with the set of forbidden confi gurations is empty, an algorithm with the following two characteristics is called 'on-the-fly':

– **Saving memory.** Computed data that are not of crucial interest for further computation are deleted from memory. That means for reachability analysis to not compute the full set of reachable confi gurations. The disadvantage of such an algorithm is that it may need more iterations and a more sophisticated condition for termination than the standard algorithm.
– **Urgent termination.** If the data computed so far allows a decision about the result then the algorithm terminates immediately. For reachability analysis this means to abort the fi xed point iteration immediately after computing some confi guration of the forbidden set. This characteristic is already implemented in the algorithm of Fig. 3. It is useful during the development of a model when it still contains errors.

For the reachability analysis it is not imperative to store the set of all reachable confi gurations. In general, it is enough to compute each reachable confi guration once to check whether it is a forbidden confi guration and to delete it from memory after this check. Our approach is to compute the sequence $Reach([\![\mathcal{A}]\!]_I)(0,0), Reach([\![\mathcal{A}]\!]_I)(1,1), \ldots$ and to keep in memory only the current set $Reach([\![\mathcal{A}]\!]_I)(i,i)$ and one of the previous sets. This strategy leads to the problem of determining if all reachable confi gurations are checked. In general, there exists no $i \in \mathbb{N}$ such that the condition $Reach([\![\mathcal{A}]\!]_C)(i,i) \supseteq Reach([\![\mathcal{A}]\!]_I)(i+1,i+1)$ holds.

A simple algorithm could detect the condition for termination by storing all computed sets $Reach([\![\mathcal{A}]\!]_I)(0,0), \ldots, Reach([\![\mathcal{A}]\!]_I)(i,i)$ and checking whether $Reach([\![\mathcal{A}]\!]_I)(i+1,i+1)$ is contained by one of the computed sets. But this would lead to very large memory overhead.

```
    Input: timed automaton $\mathcal{A} = (L, L^0, X, \Sigma, I, E)$
      with the integer semantics $[\![\mathcal{A}]\!]_I = (L \times Val(X), L^0 \times \{v^0\}, \Sigma \cup \mathbb{N}, \rightarrow)$,
      set of locations $L^F$
    Output: $true$ iff $L^F \cap ReachLoc([\![\mathcal{A}]\!]_C) \neq \emptyset$
 1  $R := L^0 \times \{v^0\}$
 2  DiscreteFixedPoint($\mathcal{A}, R$)
 3  $s := 0; p := 0$
 4  $R_{prev} := \emptyset$
 5  while $R_{prev} \not\supseteq R$
 6    if $p \leq \frac{s}{2}$
 7      $p := s$
 8      $R_{prev} := R$
 9    $R := \{q' \in L \times Val(X) \mid \exists q : q \in R \wedge q \xrightarrow{1} q'\}$
10    DiscreteFixedPoint($\mathcal{A}, R$)
11    $s := s + 1$
12    if $R \cap (L^F \times Val(X)) \neq \emptyset$ then return $true$
13  return $false$
```

Fig. 10. Algorithm for on-the-fly reachability analysis.

The algorithm given in Fig. 10 does not store all the computed sets $Reach([\![\mathcal{A}]\!]_I)(i, i)$ from previous steps but only one as R_{prev}. The computation of the fi xed point of the set R regarding only discrete transitions is abbreviated as `DiscreteFixedPoint`$(\mathcal{A}, R)$. Because this strategy stores as few information as possible and it stops immediately after fi nding an error, it is called on-the-fly analysis.

Firstly, we introduce some notation. Let $T_\mathcal{A}$ and $\Delta_\mathcal{A}$ be two natural numbers for a closed timed automaton $\mathcal{A}$ with the following properties:

1. $\Delta_\mathcal{A} > 0$,
2. $Reach([\![\mathcal{A}]\!]_I)(T_\mathcal{A}, T_\mathcal{A}) \supseteq Reach([\![\mathcal{A}]\!]_I)(T_\mathcal{A} + \Delta_\mathcal{A}, T_\mathcal{A} + \Delta_\mathcal{A})$, and
3. for all $T', \Delta' \in \mathbb{N}$ with the properties 1 and 2 the following holds:
   - $T' + \Delta' > T_\mathcal{A} + \Delta_\mathcal{A}$, or
   - $T' + \Delta' = T_\mathcal{A} + \Delta_\mathcal{A}$ and $\Delta' \geq \Delta_\mathcal{A}$.

Theorem 2. *The algorithm of Fig. 10 for on-the-fly reachability analysis terminates after less than $3(T_\mathcal{A} + \Delta_\mathcal{A})$ iterations of the while loop. If a forbidden configuration is reachable, it returns true. Otherwise, after checking all reachable configurations, it returns false.*

Proof. Due to the fi niteness of the state space (using the integer semantics), there exist $T_\mathcal{A}$ and $\Delta_\mathcal{A}$ with the properties defi ned above the theorem. After the computation of $Reach([\![\mathcal{A}]\!]_I)(0, 0), ...,$ $Reach([\![\mathcal{A}]\!]_I)(T_\mathcal{A}+\Delta_\mathcal{A}, T_\mathcal{A}+\Delta_\mathcal{A})$ all reachable confi gurations are actually checked, and the algorithm can terminate. Fig. 11 illustrates this situation. After $T_\mathcal{A}$ steps, a set of confi gurations is reached which contains (or is equal to) the set of confi gurations reachable after $T_\mathcal{A} + \Delta_\mathcal{A}$ steps. We can consider $\Delta_\mathcal{A}$ as the least common multiple of the number of steps for the corresponding loop in each trace of $\mathcal{A}$. $T_\mathcal{A}$ would be the maximal number of steps needed by the traces to enter this loop.

During each check of the condition for termination $R_{prev} \supseteq R$ in line 5, $R = Reach([\![\mathcal{A}]\!]_I)(s, s)(q_1, ..., q_n)$ and $R_{prev} = Reach([\![\mathcal{A}]\!]_I)(p, p)(q_1, ..., q_n)$ holds, p is the greatest two power number smaller than s (excepting $R_{prev} = \emptyset$ if $s = 0$; $p = 0$ if $s \leq 1$). Let k be the least two power number with $k \geq T_\mathcal{A}$ and $k \geq \Delta_\mathcal{A}$. Because of $Reach([\![\mathcal{A}]\!]_I)(k, k) \supseteq Reach([\![\mathcal{A}]\!]_I)(k+\Delta_\mathcal{A}, k+\Delta_\mathcal{A})$ the algorithm terminates at least after computation of $Reach([\![\mathcal{A}]\!]_I)(k+\Delta_\mathcal{A}, k+\Delta_\mathcal{A})$. Because $k + \Delta_\mathcal{A} < 3(T_\mathcal{A} + \Delta_\mathcal{A})$ *(1)* holds, the algorithm needs at most 3 times the number of iterations of an algorithm which stores all computed sets $Reach([\![\mathcal{A}]\!]_I)(i, i)$. If $T_\mathcal{A} = 0$ then $k < 2\Delta_\mathcal{A}$ follows and thus *(1)* holds. Otherwise from $T_\mathcal{A} > 0$ $k < 2(T_\mathcal{A} + \Delta_\mathcal{A})$ follows and *(1)* holds. □

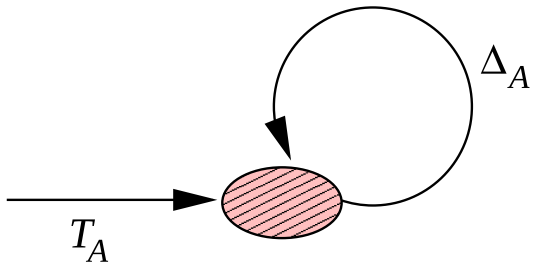

Fig. 11. Least common cycle for all traces.

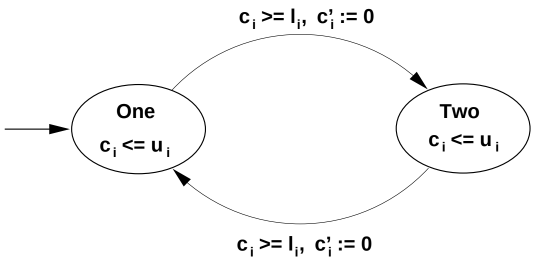

Fig. 12. The 'two state' automaton (c_i is a clock, u_i and l_i are constants).

A potential problem for the efficiency in our algorithm is that the number of iterations $T_{\mathcal{A}} + \Delta_{\mathcal{A}}$ can be considerable large in comparison with the number of iterations of the usual algorithm, which is $min\Big(\{k \in \mathbb{N} \mid Reach([\![\mathcal{A}]\!]_I)(0,k) = Reach([\![\mathcal{A}]\!]_I)(0,k+1)\}\Big) + 1$. Our experience leads to the conclusion that this strategy results in high performance for models with low explicit communication, and that it is not very detrimental for models with a lot of explicit communication.

In Table 3 we demonstrate that on-the-fly analysis can dramatically improve the efficiency. We use the little 'two state' example from [BMPY97] as presented in Fig. 12. The number of reachable configurations for the composition of several automata, each automaton with one clock, are given in the second row. The third row contains the results obtained using an on-the-fly algorithm and the fourth row indicates the explosion of the representation for the intermediate result within the computation if storing all the states (cf. the algorithm in Fig. 3). Fig. 13 visualizes this situation: the intermediate results are large BDDs. Although the components do not communicate, their configurations depend on each other because of the time transition. The on-the-fly version benefits from the fact that the configurations at a particular point in time are independent from each other (maximal BDD size is 44 nodes). The BDD-based version of Kronos needs over 20,000 seconds for 9 automata.

5.3 Refinement Check

The restricted applicability of reachability analysis due to the high time complexity of the analysis for large models leads to the need of refinement checking for verification. We implemented an algorithm for checking the existence of a simulation relation to investigate the opportunities of refinement checking for Cottbus Timed Automata [BR01]. A detailed description of these concepts is given in [Bey01a]. To confirm the practical relevance we give two examples.

Production Cell: At first we consider modular verification of the production cell described in Section 5.1. For the measurement of the throughput, i.e. how long does a piece need to go through the production cycle, we modeled each belt to be able to measure the time of transportation using a clock. For the verification process we can fade out some details of the machines. To verify a safety property, e.g. 'the drilling machine must be off if the transport belt is not off', we verify at first that the timed version of the transport belt implements an untimed version by checking the existence

Number of components	4	8	16	32	64
Number of reachable configurations	$3.3 \cdot 10^5$	$1.1 \cdot 10^{11}$	$1.2 \cdot 10^{22}$	$1.5 \cdot 10^{44}$	$2.2 \cdot 10^{88}$
On-the-fly computation	2.2	4.5	11	30	94
Storing all computed configurations	3.0	170			

Table 3. Time for computation of all reachable configurations of the **two state** example (for $u_i = 12, l_i = 9$).

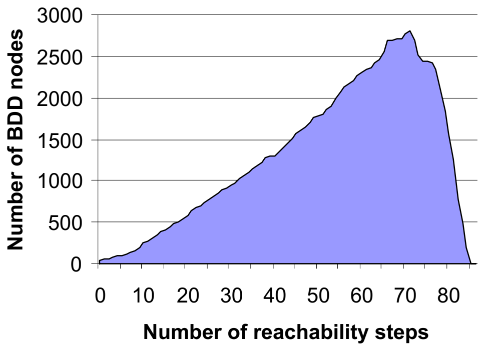

Fig. 13. The intermediate BDD size while computing the reachable set for 4 **two state** automata.

of a simulation relation. Now we can verify the safety property of the model using that smaller untimed version for transport belts. Table 4 compares the measurements for the following concrete verification tasks: In the first experiment we analyzed the safety property of the system using a timed model for the sensor instances. In the second experiment we analyzed the system using an untimed version for the sensors. It shows that an abstraction within one small part of the system has a big impact on the computation time. The last row presents the computation time for the simulation check. Because the sensor model is a small part of the whole system this verification task do not need much time.

Simple mutex protocol: Another example is the verification of a very simple protocol for mutual exclusion. Each process has three states: *uncritical*, *wait* and *critical*. Going from *uncritical* to *wait* it sends a signal *announce* to its scheduler. The scheduler has to decide whether the process can use the exclusive resource. If yes, then the scheduler sends the signal *acknowledge* to the process. Now the process can use the resource in its critical section. Sending a signal *release* to the scheduler the process frees the resource. Each scheduler controls (encapsulated) two processes, and the scheduler itself behaves exactly like a process to its environment. Thus, we can build up a tree consisting of schedulers (nodes) and processes (leafs).

For the verification of the mutual exclusion property we could verify the product automaton for the flattened composition using reachability analysis, but it becomes unfeasable using a lot of processes. Applying an inductive proof, we can verify the mutual exclusion property for an initial number of processes using reachability analysis (start of induction). For one process it is trivial even without any tool. Assuming that the property is fulfilled for n processes (inductive hypothesis) we can conclude that the property is also fulfilled using $2n$ processes (inductive step). For the inductive step we use the existence of a simulation relation between the scheduler and the process, i.e. the scheduler implements the process. The verification of 'SchedulerWith2Processes' refines 'OneProcess' needs 0.5 seconds computation time.

Verification task	Computation time
System using 'TimedSensor' to model sensors	1098
System using 'UntimedSensor' to model sensors	556
'TimedSensor' refines 'UntimedSensor'	0.5

Table 4. Verification of a safety property for the **production cell**.

6 Summary

In extension of [ABK+97,AMP98] we gave a formal definition of an integer semantics for closed timed automata and proved the correctness of using this semantics for reachability analysis. Based on the integer semantics we developed a tool implementation which uniformly represents locations and clock assignments as BDDs [Bey01c]. The tool applies additional BDD-related techniques, on-the-fly analysis and refinement checking for further performance improvements. Our experiments suggest that it computes the reachability set in polynomial time and space for a particular class of models. To get good variable orderings we use a size estimate for the set of reachable configurations [Bey01b].

Acknowledgments

We thank Claus Lewerentz and Heinrich Rust for critical discussions and valuable hints for improvement of the paper.

References

[ABK+97] Eugene Asarin, Marius Bozga, Alain Kerbat, Oded Maler, Amir Pnueli, and Anne Rasse. Data-structures for the verification of timed automata. In O. Maler, editor, *Proceedings of the 1st International Workshop on Hybrid and Real-Time Systems (HART'97)*, LNCS 1201, pages 346–360. Springer-Verlag, 1997.

[Alu99] Rajeev Alur. Timed automata. In N. Halbwachs and D. Peled, editors, *Proceedings of the 11th International Conference on Computer-Aided Verification (CAV'99)*, LNCS 1633, pages 8–22. Springer-Verlag, 1999.

[AMP98] Eugene Asarin, Oded Maler, and Amir Pnueli. On discretization of delays in timed automata and digital circuits. In R. de Simone and D. Sangiorgi, editors, *Proceedings of the 9th International Conference on Concurrency Theory (CONCUR'98)*, LNCS 1466, pages 470–484. Springer-Verlag, 1998.

[ATB94] Adnan Aziz, Serdar Tasiran, and Robert K. Brayton. BDD variable ordering for interacting finite state machines. In *Proceedings of the 31st ACM/IEEE Design Automation Conference (DAC'94)*, pages 283–288, 1994.

[BDM+98] Marius Bozga, Conrado Daws, Oded Maler, Alfredo Olivero, Stavros Tripakis, and Sergio Yovine. Kronos: a model-checking tool for real-time systems. In A.J. Hu and M.Y. Vardi, editors, *Proceedings of the 10th International Conference on Computer-Aided Verification (CAV'98)*, LNCS 1427, pages 546–550. Springer-Verlag, 1998.

[Bey01a] Dirk Beyer. Efficient reachability analysis and refinement checking of timed automata using BDDs. In *Proceedings of the 11th Advanced Research Working Conference on Correct Hardware Design and Verification Methods (CHARME 2001, Livingston)*, to appear, LNCS. Springer-Verlag, 2001.

[Bey01b] Dirk Beyer. Improvements in BDD-based reachability analysis of timed automata. In Jose Nuno Oliveira and Pamela Zave, editors, *Proceedings of the 10th International Symposium of Formal Methods Europe (FME 2001, Berlin): Formal Methods for Increasing Software Productivity*, LNCS 2021, pages 318–343. Springer-Verlag, 2001.

[Bey01c] Dirk Beyer. Rabbit: Verification of real-time systems. Technical Report I-05/2001, BTU Cottbus, 2001.

[BMPY97] Marius Bozga, Oded Maler, Amir Pnueli, and Sergio Yovine. Some progress on the symbolic verification of timed automata. In O. Grumberg, editor, *Proceedings of the 9th International Conference on Computer Aided Verification (CAV'97)*, LNCS 1254, pages 179–190. Springer-Verlag, 1997.

[BR00] Dirk Beyer and Heinrich Rust. A tool for modular modelling and verification of hybrid systems. In Alfons Crespo and Joan Vila, editors, *Proceedings of the 25th IFAC/IFIP Workshop on Real-Time Programming 2000 (WRTP 2000, Palma)*, pages 169–174. Elsevier Science, Oxford, 2000.

[BR01] Dirk Beyer and Heinrich Rust. Cottbus Timed Automata: Formal definition and semantics. In Charles Rattray, Miroslav Sveda, and Jerzy Rozenblit, editors, *Proceedings of the 2nd IEEE/IFIP Joint Workshop on Formal Specifications of Computer-Based Systems (FSCBS 2001, Washington, D.C.)*, pages 75–87, Stirling, 2001.

[Bry86] Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transaction on Computers*, C-35(8):677–691, 1986.

[BW96] Beate Bollig and Ingo Wegener. Improving the variable ordering of OBDDs is NP-complete. *IEEE Transactions on Computers*, 45(9):993–1002, September 1996.

[Dil89] David L. Dill. Timing assumptions and verification of finite-state concurrent systems. In J. Sifakis, editor, *Proceedings of the International Workshop on Automatic Verifi cation Methods for Finite State Systems*, LNCS 407, pages 197–212. Springer-Verlag, 1989.

[GPV94] Aleks Göllü, Anuj Puri, and Pravin Varaiya. Discretization of timed automata. In *Proceedings of the 33rd IEEE Conference on Decision and Control*, pages 957–958, 1994.

[HMP92] Thomas A. Henzinger, Zohar Manna, and Amir Pnueli. What good are digital clocks? In *Proceedings of the 19th International Colloquium on Automata, Languages, and Programming (ICALP'92)*, LNCS 623, pages 545–558. Springer-Verlag, 1992.

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

[LL95] Claus Lewerentz and Thomas Lindner, editors. *Formal Development of Reactive Systems*. LNCS 891. Springer-Verlag, Berlin, Heidelberg, 1995.

[LLKS85] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys, editors. *The Traveling Salesman Problem*. John Wiley & Sons, 1985.

[LPY95] Kim G. Larsen, Paul Pettersson, and Wang Yi. Compositional and symbolic model-checking of real-time systems. In *Proceedings of the 16th IEEE Real-Time Systems Symposium (RTSS'95)*, pages 76–87, 1995.

[LPY97] Kim G. Larsen, Paul Pettersson, and Wang Yi. UPPAAL in a Nutshell. *International Journal on Software Tools for Technology Transfer*, 1(1-2):134–152, October 1997.

[RAB$^+$95] Rajeev K. Ranjan, Adnan Aziz, Robert K. Brayton, Carl Pixley, and Bernhard Plessier. Efficient BDD algorithms for synthesizing and verifying finite state machines. In *Workshop Notes of the IEEE/ACM International Workshop on Logic Synthesis (IWLS'95)*, 1995.

[Wan00] Farn Wang. Efficient data structure for fully symbolic verification of real-time software systems. In S. Graf and M. I. Schwartzbach, editors, *Proceedings of the 6th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2000)*, LNCS 1785, pages 157–171. Springer-Verlag, 2000.