Rabbit: Verification of Real-Time Systems

Dirk Beyer

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

Abstract. This paper gives a short overview of a model checking tool for Cottbus Timed Automata, which is a modular modeling language based on timed and hybrid automata. For timed automata, the current version of the tool provides BDD-based verification using an integer semantics. Reachability analysis as well as refinement checking is possible. To find good variable orderings it uses the component structure of the model and an upper bound for the BDD size. For hybrid automata, reachability analysis based on the double description method is implemented.

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

1 Introduction

Nowadays, software engineering research of real-time systems has to investigate methods for formal specification and verification. Our research group works on formal methods and tries to define development processes for systems like production cells or real-time algorithms. As basic formalisms we chose *hybrid automata* [Hen96] and *timed automata* [AD94] because they have a well-founded theoretical basis. Reachability analysis of such models has been implemented in several tools, e.g. HyTech, Kronos and Uppaal. From our point of view, two major problems are the lack of concepts for modeling large systems and the exploding consumption of time and memory by the verification algorithms. We address these issues with our tool *Rabbit* providing the following features:

- **Modular modeling** using an extension of timed and hybrid automata called Cottbus Timed Automata (CTA) [BR98]. Automata describing the behavior of the system are encapsulated by modules. Communication with other modules is possible via shared variables and synchronization labels declared within an *interface*. Each interface component has a particular access mode (read only, exclusive write, etc.) to restrict the use of variables and synchronization labels. Replicated subsystem components do not have to be multiply defined. They are instantiated from a common *template module*. Subsystem descriptions can be grouped to build hierarchical structures. An extension of the formalism leads to a *compositional semantics*, i.e. we can define the semantics of a CTA module on the basis of the semantics of its components [BR01].
- **Reachability analysis.** The tool provides efficient reachability analysis for timed automata using a BDD representation based on an integer semantics. We use the modular structure to compute variable orderings allowing for efficient representation of the transition relation (resp. the set of reachable configurations). The reachability analysis for hybrid automata is based on the double description method (DDM) [FP96].
- **Refinement checking.** To make verification of large systems tractable, the notation allows to replace specific modules by more abstract versions. To prove the correctness of this replacement, we have to check whether two modules have the same behavior respecting external synchronization labels. The tool checks the refinement relation by checking the existence of a simulation relation.
- **Framework architecture.** The tool is built following the idea to have a platform for model checking of timed as well as hybrid systems. It is capable to use other representations, also from foreign libraries. Currently, the tool chooses dynamically the BDD representation if the model consists of (closed) timed automata. Otherwise, having a model consisting of hybrid automata, the tool uses the DDM library.

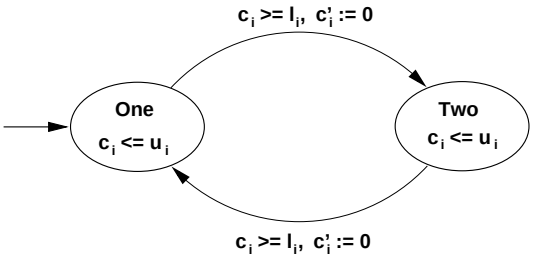

Fig. 1. The 'two state' automaton (c_i is a clock, u_i and l_i are constants).

– **Representation libraries.** To get fast BDD operations we implemented our own library without any additional overhead like general purpose BDD libraries. We implemented a library for the Double Description Methods (DDM) representation [FP96] similar to the one used by HyTech.

Other existing approaches, e.g. MOCHA, CHARON and MASACCIO, address similar issues but are based on modeling formalisms different from timed automata. MOCHA is based on *reactive modules* (and partially on *timed modules*). It provides refinement checking, but only for the untimed case [AHM+98]. CHARON is a modeling language based on hierarchic hybrid modules [AGH+00]. Currently, there is no model checking tool available for CHARON models. MASACCIO is a formal model for hybrid dynamical systems for modeling continuous components based on differential equations [Hen00].

2 Reachability Analysis

Safety properties can be verified by model checking, especially reachability analysis. The main problem is the exploding consumption of time for the computation and memory for the representation of the reachable configurations. 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 over d clocks which are subsets of $\mathbb{R}_+^d$. For the symbolic representation of sets of *locations* binary decision diagrams (BDDs) are widely used. To get a uniform representation of locations and clock assignments as BDDs we formally defined an *integer semantics* of so-called *closed timed automata*, whose clock constraints do not contain the relations $<$ and $>$. Thus, we need to consider only integer clock assignments. We proved its location equivalence (i.e. the sets of reachable *locations* are equivalent) to the usual, continuous semantics [Bey01b]. Proofs of the location equivalence using integer semantics for models different from timed automata can be found in literature [Pop91,HMP92,AMP98].

A discretization of time which is location equivalent to the continuous semantics exists for all timed automata [GPV94]. They use fractions of 1 as time steps which depend on the number of clocks within the model. It means that the time step is very small for a large number of clocks and thus, there is a very large state space. For such a discrete semantics, [ABK+97] contains a proof of the location equivalence to the continuous semantics.

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.

In the context of this paper, we give only a rough illustration of our modeling notation. Figure 2 is the textual representation of two automata from Figure 1. The modular structure of the AND model, which is more representative for the instantiation mechanism, is given in [Bey01b].

Usually, the transition relation is represented as implicit union of a timed transition relation and discrete transition relations for each synchronization label. Our experiments have confirmed that

```
MODULE TwoState {
  INPUT
    l:            CONST;
    u:            CONST;
  INITIAL   STATE(twoState) = one  AND  c = 0;

  AUTOMATON twoState {
    STATE one  { INV   c + 1 <= u;
                 TRANS { GUARD   c >= l;
                         DO   c' = 0;
                         GOTO two;
                       }
               }
    STATE two  { INV   c + 1 <= u;
                 TRANS { GUARD   c >= l;
                         DO   c' = 0;
                         GOTO one;
       }
               }
  }
  LOCAL
    c:            CLOCK(13);  // (u + 1)
}

MODULE System {
  LOCAL
    l = 9       : CONST;
    u = 12      : CONST;
  INST automaton1 FROM TwoState WITH {
    l           AS l;
    u           AS u;
  }
  INST automaton2 FROM TwoState WITH {
    l           AS l;
    u           AS u;
  }
}
```

Fig. 2. The textual representation of the 'two state' model.

applying such **partial transition relations** sequentially is more efficient than using the union of these relations as monolithic transition relation [RAB+95].

When using several partial transition relations, we have to determine the **order of** their **application**. The intermediate sets of reached configurations in the reachability algorithm 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.

Variable Ordering. Using the BDD representation we have to find good variable orderings. We start with an initial ordering and then we apply a heuristic to increase the quality of the ordering. We take the pre-order linearization (parent node first, then its children together, recursively applied) of the CTA model as initial variable ordering. This implies that we consider the modeler's decision to encapsulate some components together within one module, i.e. local components of a module are assigned to neighboring positions within the variable ordering. Then we apply another heuristic, which optimizes the ordering respecting an estimate for the size of the BDD representing the reachable set of configurations. This estimate is based on an upper bound for the size of the BDD for the transition relation [Bey01b]. It reflects the two most important characteristics for good variable orderings: **(1)** Communicating components have neighboring positions within the ordering. **(2)** Components which communicate with many other components precede these other components within the ordering. In difference to the most existing BDD applications we prefer to use that static ordering instead of dynamic reordering, which consumes a lot of run-time. Compared with the BDD-based version of Kronos [BMPY97] and Wang's tool [Wan00], our strategy for variable ordering leads to a significant performance improvement.

We illustrate the dramatic influence of the variable ordering on the size of the BDDs representing the reachable set. Figure 3 gives the shape of the BDD for the set of all reachable configurations for

8 Fischer processes. The large one (light grey) uses the ordering 'k at last' (which invalidates one of the characteristics for good orderings) and the small one (dark grey) uses 'k in front'. 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 horizontal line represents a BDD level with a lot of BDD nodes.

Fig. 3. The BDD shape for the set of all reachable configurations for two different variable orderings (**Fischer8**).

For a model with a simple communication structure like Fischer's protocol it might be obvious to obtain good variable orderings. To motivate the need for automatic, estimate-based variable ordering we give another example: Figure 4 shows two BDDs for the reachable set using a production cell model which consists of 45 timed automata with 22 clocks. In the first experiment we used the variable ordering regarding the modular structure of the model, but the estimate-based heuristics 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, light shape). If we apply our heuristics, we get a significantly better variable ordering (small, dark shape). This estimate-based computation of the variable ordering is done automatically by our tool.

Fig. 4. The BDD shape for the full reachable set for two different variable orderings for the **production cell** model.

3 Refinement Checking

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 capabilities of refinement checking for Cottbus Timed Automata.

Language inclusion is undecidable for timed automata [AD94]. Although it is decidable for closed timed automata, trace inclusion is of intractable time complexity. Therefore, we use a simulation relation for the algorithmic analysis within our tool implementation. This is justified by the assumption that the two modules between which the refinement relation should exist have a similar structure, which is mostly fulfilled using stepwise refinement within the development process.

A labeled transition system Q simulates a labeled transition system P if Q can match every step of P by a step with the same label. We use the concept of safety simulation relation as described in [DHWT92].

The algorithm of our simulation check works as follows: Firstly, we compute the composition for P and Q. Secondly, we compute the set of reachable configurations of this composition. We consider this set of tuples (p, q) as the initial relation for trying to build a simulation relation between P and Q. Then, in each cycle of a fixed point iteration we assume that it is the simulation relation and we check whether all configurations of the set are fulfilling the simulation condition mentioned above. If there are 'bad' configurations we have to invalidate our assumption that it is already the simulation relation and we eliminate them from the relation. If we reached the fixed point (i.e. our assumption was true), we got the simulation relation.

More details about our implementation of refinement checking are given in [Bey01a].

4 Verification Results

To demonstrate the performance of Rabbit, we examine the verification task for some example models.

4.1 Reachability Analysis

Fischer's protocol. For Fischer's protocol for timing-based mutual exclusion (the simplified version which permits only one process to enter the critical section) we verified the mutual exclusion property for n processes. The first and second row in Table 1 show the computation times we obtained using publicly available versions of Kronos and Uppaal. These tools use difference bound matrices to represent sets of clock assignments, and thus, the computation times seem to be at least exponential in n, while the computation time of our tool seems to be polynomial using a good variable ordering (third row). A BDD-based version of Kronos is able to verify 14 processes as reported in [BMPY97], which also looks like exponential growth of computation time. Better results (than in the table) for Uppaal are reported in [LPY95], but still exponential (20 seconds for 6 processes, 150 seconds for 7 processes). Wang reports verification of 17 processes in 15,330 seconds on a Pentium II with 366 MHz and 256 MB memory [Wan00]. Figure 5 compares Rabbit with Wang's tool, which uses a BDD-like data structure called DDD.

No. processes	4	5	6	7	8	10	12	14	16	32	64	128
Kronos	3.0	191	MO									
Uppaal	0.5	13.0	657	MO								
Rabbit	0.3	0.4	0.8	1.3	2.3	4.0	8.9	13.6	22.7	208	1920	12684

Table 1. Computation time for verification of Fischer's protocol, given in seconds of CPU time on a SUN Ultra-Sparc 1 with 200 MHz processor. 'MO' means memory overflow.

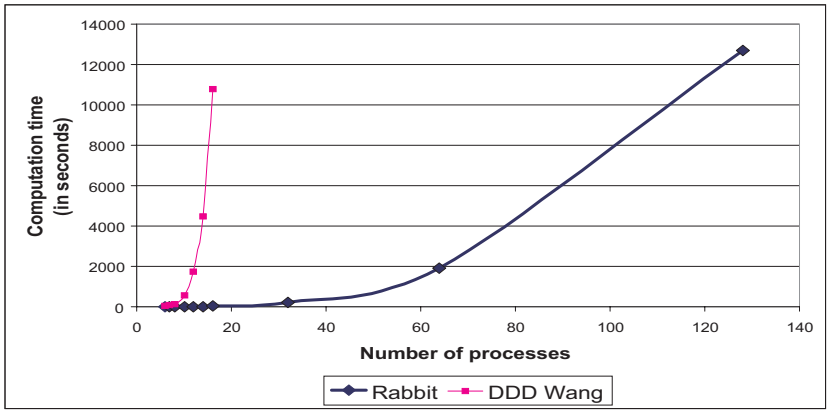

Fig. 5. Computation times for the verification of the mutex property (**Fischer's protocol**).

AND circuit. Table 2 shows the results applying our tool to compute the whole set of reachable configurations of the AND model. The BDD-based version of Kronos needs 324.7 seconds for the AND model with 4 inputs, also obtained using a SUN Ultra-Sparc 1 [BMPY97].

Number of input signals	2	4	8	16
Rabbit	0.5	6.0	79.6	1208.7

Table 2. Time for computation of all reachable configurations of the AND model, given in seconds of CPU time on a SUN Ultra-Sparc 1 with 200 MHz processor.

Two state automata. In Table 3 we demonstrate that not only the variable ordering but also the strategy of the algorithm is of crucial importance for the efficiency. We use the little 'two state' example from [BMPY97] as presented in Figure 1. The number of reachable configurations for the composition of several automata, each automaton with one clock, are given in the first row. The second row contains the results obtained using our on-the-fly algorithm and the third row indicates the explosion of the representation for the intermediate results within the computation if storing all the states. On-the-fly computation means to check all reachable configurations without storing the whole set.

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

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

Figure 6 indicates the fast growing of the representation for the intermediate sets (i.e. the largest intermediate BDD before a time transition is taken) within the computation if storing all the configurations. Although the components have no communication between each other, their configurations depend on each other because of the time transition. The on-the-fly version benefits from the fact that the configurations on a particular point in time are independent from each other (maximal BDD

size is less than 100 nodes, cf. Figure 7). We have to refer to a more formal paper for a detailed description and the proof of termination for our on-the-fly algorithm [BN01].

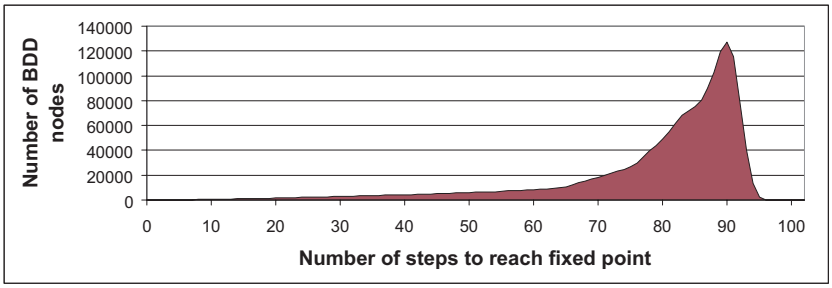

Fig. 6. The number of nodes of the BDD during the *full-set computation* (**TwoState8**).

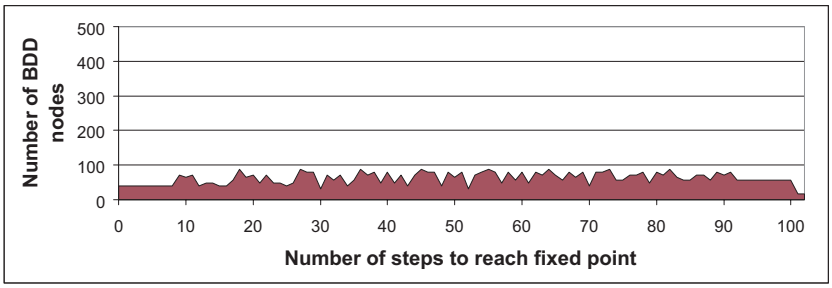

Fig. 7. The number of nodes of the BDD during the computation of the reachable configurations using *on-the-fly computation* (**TwoState8**).

4.2 Refinement Check

Production cell. To validate the practical relevance of our tool using a more realistic case study, we developed a CTA model of a production cell, which is similar to the Lewerentz/Lindner production cell [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.

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 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 does not need much time.

Simple mutex protocol. For illustration we use the verification of a very simple protocol for mutual exclusion. Each process has three states: *uncritical*, *wait* and *critical*. Going from *uncritical*

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**.

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 children (either a process or a scheduler again), 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 two children (inductive hypothesis) we can conclude that the property is also fulfilled for the scheduler that contains these two children (inductive step). For the inductive step we use the existence of a simulation relation between the scheduler and the children, i.e. the scheduler implements the child. The verification that 'SchedulerWith2Processes' refines 'OneProcess' needs 0.5 seconds computation time.

4.3 Status of the project

The tool implementation of reachability analysis is completed for both representations, DDM and BDD. The current version of Rabbit contains refinement checking for (closed) timed automata. The objective of the flexible architecture is to serve as a framework for further exploration of algorithms and data structures belonging to verification of hybrid and timed systems. The tool Rabbit and the related papers are available from `http://www-sst.informatik.tu-cottbus.de/Rabbit`.

Because of the lack of examples reflecting real-world systems, we built a Fischer-Technik model of a production cell consisting of 20 machines and belts with 44 sensors and 28 motors. Our work is now focused on validating different verification methods for the CTA model of this system. Another question within this context is whether it is possible to analyze 'large' hybrid systems in a way that we at first derive a timed version of the model by proving the refinement relation between timed and hybrid modules to get a more abstract model. Then we can use the efficient analysis of (closed) timed automata or we have to do further abstraction steps. An open research task is to investigate efficient representations combining the advantages of BDDs with the advantages of difference bound matrices (DBM).

Acknowledgments

We thank Andreas Noack for his valuable work within his diploma thesis, and we thank Heinrich Rust and Claus Lewerentz for discussions.

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.

[AD94] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[AGH+00] R. Alur, R. Grosu, Y. Hur, V. Kumar, and I. Lee. Modular specification of hybrib systems in CHARON. In *Proceedings of the 3th International Workshop on Computation and Control (HSCC 2000)*, Pittsburgh, PA, 2000.

[AHM+98] R. Alur, T.A. Henzinger, F.Y.C. Mang, S. Qadeer, S.K. Rajamani, and S. Tasiran. MOCHA: Modularity in model checking. In *Proceedings of the 10th International Conference on Computer-aided Verification (CAV 1998)*, LNCS 1427, pages 521–525. Springer-Verlag, 1998.

[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.

[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)*, LNCS 2144. Springer-Verlag, Berlin, 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, Berlin, 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.

[BN01] Dirk Beyer and Andreas Noack. Efficient verification of timed automata using BDDs. In Stefania Gnesi and Ulrich Ultes-Nitsche, editors, *Proceedings of the 6th International ERCIM Workshop on Formal Methods for Industrial Critical Systems (FMICS 2001, Paris)*, pages 95–113. INRIA, Paris, 2001.

[BR98] Dirk Beyer and Heinrich Rust. Modeling a production cell as a distributed real-time system with Cottbus Timed Automata. In Hartmut König and Peter Langendörfer, editors, *Tagungsband Formale Beschreibungstechniken für verteilte Systeme (FBT 1998, Cottbus)*, pages 148–159. Shaker Verlag, Aachen, 1998.

[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.

[DHWT92] David L. Dill, Alan J. Hu, and Howard Wong-Toi. Checking for language inclusion using simulation preorders. In A. Skou K. Larsen, editor, *Proceedings of the 3rd International Workshop on Computer Aided Verification (CAV'91)*, LNCS 575, pages 255–265. Springer-Verlag, 1992.

[FP96] Komei Fukuda and Alain Prodon. Double description method revisited. In *Combinatorics and Computer Science*, LNCS 1120, pages 91–111. Springer-Verlag, 1996.

[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.

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

[Hen00] Thomas A. Henzinger. Masaccio: a formal model for embedded components. In *Proceedings of the First IFIP International Conference on Theoretical Computer Science (TCS2000)*, LNCS 1872, pages 549–563. Springer-Verlag, 2000.

[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.

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

[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.

[Pop91] Louchka Popova. On time petri nets. *Journal of Information Processing, EIK*, 27(4):227–244, 1991.

[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.