Using Interval Diagram Techniques for the Symbolic Verification of Timed Automata

Karsten Strehl

Computer Engineering and Networks Lab (TIK)
Swiss Federal Institute of Technology (ETH)
Gloriastrasse 35, 8092 Zurich, Switzerland
eMail: strehl@tik.ee.ethz.ch
WWW: http://www.tik.ee.ethz.ch

TIK Report
No. 53, July 1998

Abstract

In this report, we suggest *interval diagram techniques* for formal verification of *timed automata*. Interval diagram techniques are based on *interval decision diagrams* (IDDs)—representing sets of system configurations of, e.g., timed automata—and *interval mapping diagrams* (IMDs)—modeling their transition behavior. IDDs are canonical representations of Boolean functions and allow for their efficient manipulation. We present the methods necessary for our approach and compare its results to another, similar verification technique.

Contents

Chapter 1

Introduction

Especially for safety-critical applications like those in traffic control, medical engineering, or avionics, simulation often is not sufficient to guarantee the correctness of a technical system's model. Additionally, formal methods are employed to verifiy the system behavior and to determine timing properties. Several approaches exist to model timing behavior, mostly derived from conventional finite state automata which are expanded to describe timing properties of the transition behavior. As one of the most universal ones, Alur and Dill have proposed *timed automata* [AD94], represented by state-transition graphs with timing constraints using finitely many clocks.

In [MP95], Maler and Pnueli describe a possible application of timed automata for modeling asynchronous circuits. The digital circuit considered is transformed into a timed automaton reflecting timing behavior aspects such as uncertainties in gate delays and input arrival times. The constructed automaton may be used for formal verification or timing analysis. Besides reachability analysis, formal verification comprises real-time symbolic model checking, i.e., checking the satisfaction of timing properties expressed in one of various real-time temporal logics. Timing constraints on input signals may be inferred or delay characteristics required in order to meet some given behavioral specifications may be calculated.

The most severe restrictions on formal verification of timed automata result from resource limitations, i.e., computational power and memory. Formal verification such as reachability analysis of timed automata may be performed using *difference bounds matrices* (DBMs) [Dil89] to represent *clock regions* during computation, as explained later on. As DBM methods often fail for large models, other approaches have been proposed using different kinds of region representations. For instance, *numerical decision diagrams* (NDDs) [ABK+97, BMPY97], a derivative of *binary decision diagrams* (BDDs) [Bry86], have been employed successfully.

Interval diagram techniques—using *interval decision diagrams* (IDDs) and *interval mapping diagrams* (IMDs)—have shown to be convenient for formal verification of, e.g., process networks [ST98a] or Petri nets [ST98b], often providing advantages regarding computation time and memory resources. In this report, interval diagram techniques are applied to formal verification of timed automata. We present the used interval diagrams and verification techniques and compare their runtime behavior with that of the NDD approach. In Section 2, timed automata and their analysis are summarized. We briefly present IDDs and IMDs in Section 3. Section 4 explains how formal verification of timed automata may be performed especially using interval diagram techniques, while Section 5 presents experimental results concerning this. Finally, Section 6 gives a short summary.

Chapter 2

Timed Automata

Figure 2.1 shows an example timed automaton. It may be used to model two independent non-deterministic input oscillators of which the pulse widths are known only in certain time ranges. First, we give a brief and informal introduction to timed automata. They will be defined formally later on.

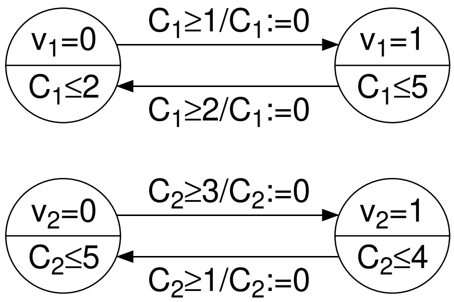

Figure 2.1: Example timed automaton.

The automaton of Figure 2.1 has four *locations* depicted by circles and two clocks C_1 and C_2 which we suppose to be set to 0 at the beginning. The product of the locations of the partial automata results in four discrete states $q \in Q$ with $q = (v_1, v_2)$ and $Q = \big\{(0,0),(0,1),(1,0),(1,1)\big\}$. Starting with the *configuration* $(q, C_1, C_2) = \big((0,0),0,0\big)$, representing the entity of discrete state and all clock values, time progresses and makes the values of C_1 and C_2 increase uniformly. The automaton is allowed to stay in a certain location as long as the corresponding *staying condition*—depicted in the lower part of each location—is satisfied. The *guards* at the transitions represent conditions which have to be fulfilled to enable the respective transition. If a transition is taken, given clocks are reset to 0.

2.1 The Timed Automaton

Timed automata are completely defined and described in [AD94]. In this section, we use the following definitions analogous to [ABK$^+$97]. Bold-face letters are used to denote vectors in $\mathbb{R}^d$, i.e., $\mathbf{v}$ stands for $(v_1, \ldots, v_d)$ where $v_i \in \mathbb{R}$ for $i = 1, \ldots, d$. For $\mathbf{u}, \mathbf{v} \in \mathbb{R}^d$, $\mathbf{u} \leq \mathbf{v}$ denotes that $u_i \leq v_i$ for $i = 1, \ldots, d$. A set $S \subseteq \mathbb{R}^d$ is said to be *monotonic* iff for every $\mathbf{u} \in \mathbb{R}^d$ satisfying $\mathbf{u} \leq \mathbf{v}$, $\mathbf{v} \in S$ implies $\mathbf{u} \in S$.

$G_{qq'}$ denotes the subset of the clock space satisfying the transition guard from q to $q' \neq q$, while G_{qq} represents the set of clock values satisfying the staying condition of q. The number of clocks is denoted by d. For timed automata, $G_{qq'}$ and G_{qq} are restricted to be *k-polyhedral* subsets of $\mathbb{R}^d$—sets resulting from the application of set-theoretic operations to half-spaces of the form $\{\mathbf{v} : v_i \leq c\}$, $\{\mathbf{v} : v_i < c\}$, $\{\mathbf{v} : v_i - v_j \leq c\}$, or $\{\mathbf{v} : v_i - v_j < c\}$ for some integer $c \in \{0, \ldots, k\}$. Such sets are called *regions* and constitute the *region graph* [AD94] of which the properties underlie all analysis methods for timed automata. $R_{qq'} : \mathbb{R}^d \to \mathbb{R}^d$ is the *reset function* associated with q and q', setting some of its arguments to 0 while leaving the others intact.

Without loss of generality, the timed automata considered are restricted as follows:

1. At most one transition is associated with every pair of locations,
2. the clock space is $[0, k)^d$ as the clock values are bounded by k,
3. $G_{qq'}$ is convex for every $q, q' \in Q$, and
4. G_{qq} is monotonic for every $q \in Q$.

Any timed automaton may be easily transformed into one satisfying these properties.

K denotes the interval $[0, k)$ for dense time or the set $\{0, \ldots, k-1\}$ for discrete time. $\mathbf{z} + t$ stands for $\mathbf{z} + t \cdot \mathbf{1}$, where $\mathbf{1} = (1, \ldots, 1)$ is the d-dimensional unit vector.

Definition 2.1.1 (Timed Automaton) *A timed automaton is a triple $\mathcal{A} = (Q, Z, \delta)$ such that*

- *Q is a discrete* state set*,*
- *$Z = K^d$ is the* clock space *($Q \times Z$ is the* configuration space*), and*
- *$\delta : Q \times Z \to 2^{Q \times Z}$ is the* transition relation *admitting the following decomposition: For every $q, q' \in Q$, let $G_{qq'} \subseteq Z$ be a k-polyhedral monotonic set and let $R_{qq'} : Z \to Z$ be a reset function. Then, for every configuration $(q, \mathbf{z}) \in Q \times Z$,*

$$\delta(q, \mathbf{z}) = \Big\{(q', \mathbf{z}') : \exists t \in K \text{ such that } (\mathbf{z} + t \in G_{qq} \cap G_{qq'}) \wedge \big(\mathbf{z}' = R_{qq'}(\mathbf{z} + t)\big)\Big\}. \tag{2.1}$$

2.2 Time Forward Projection

The application of the transition relation $\delta(q, \mathbf{z})$ results in the set consisting of all configurations reachable from $(q, \mathbf{z})$ after waiting some time t (which may be zero) and then taking at most one transition. The process of waiting before the possible discrete transition is called *time forward projection* and defined as a function $\Phi : 2^Z \to 2^Z$ with

$$\Phi(P) = \{\mathbf{z} + t : \mathbf{z} \in P, t \in K\} \cap Z. \tag{2.2}$$

(q, P) denotes subsets of $Q \times Z$ of the form $\{q\} \times P$ where P is k-polyhedral. All subsets of $Q \times Z$ encountered in the analysis of timed automata are decomposable into a finite union of such sets. Functions on elements are extended to functions on sets in the natural way, e.g., $\delta\big((q, P)\big) = \bigcup_{\mathbf{z}\in P} \delta(q, \mathbf{z})$ and $R_{qq'}(P) = \bigcup_{\mathbf{z}\in P} R_{qq'}(\mathbf{z})$.

With $P^\Phi = \Phi(P) \cap G_{qq}$ and $P_{q'} = R_{qq'}(P^\Phi \cap G_{qq'})$ for every q', the immediate successors of a set of configurations (q, P) are denoted as

$$\delta\big((q, P)\big) = (q, P^\Phi) \cup \bigcup_{q' \neq q} (q', P_{q'}). \tag{2.3}$$

Figure 2.2 shows the configurations reachable after up to one transition of each partial automaton of Figure 2.1. The state $q = (v_1, v_2)$ corresponding to each of the three regions is given. Beginning with the initial state $q^0 = (0, 0)$ and configuration $(q, C_1, C_2) = (q^0, 0, 0)$, time is projected resulting in the oblique line in the figure, restricted by $C_1 \le 2$. This corresponds to the time forward projection $\Phi(P^0) \cap G_{q^0 q^0}$ of the initial clock region $P^0 = \big\{(C_1, C_2) : C_1 = C_2 = 0\big\}$, restricted by the staying condition of q^0. For $C_1 \ge 1$, C_1 may be reset to 0 by taking the transition from $v_1 = 0$ to $v_1 = 1$. The result of this is depicted by the bold line at the left border of the dark-shaded trapezoid. The set of immediate successors $\delta\big((q^0, P^0)\big)$ consists of the union of the sets of configurations corresponding to both lines mentioned above. Projecting time up to $C_2 \le 5$ yields the complete trapezoid for $(1, 0)$. Resetting C_2 for $C_2 \ge 3$ and projecting time results in the light-shaded trapezoid for $(1, 1)$.

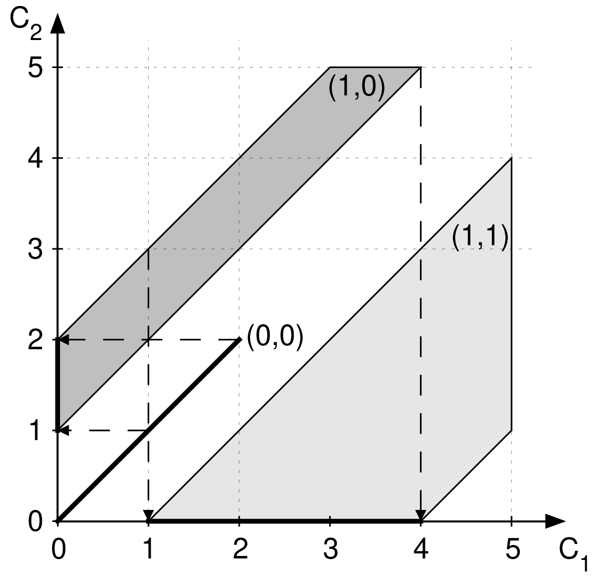

Figure 2.2: Configurations reachable first.

Chapter 3

Interval Diagram Techniques

For formal verification of, e.g., process networks [ST98a] and Petri nets [ST98b], interval diagram techniques—using interval decision diagrams (IDDs) and interval mapping diagrams (IMDs)—have shown to be a favorable alternative to BDD techniques. This results from the fact that for this kind of models of computation, the transition relation has a very regular structure that IMDs can conveniently represent. While BDDs have to represent explicitly all possible state variable value pairs before and after a certain transition, IMDs store only the *state distance*—the difference between the state variable values before and after the transition. Especially for models with large numbers of tokens, this approach is reasonable and useful. IDDs are used to represent state sets during computations. IDDs and IMDs are presented in detail in [ST98b]. In this report, we only give a brief, informal summary of their structure and properties and the methods required.

3.1 Interval Decision Diagrams

IDDs are a generalization of BDDs and MDDs—*multi-valued decision diagrams* [SKMB90]—allowing diagram variables to be integers and child nodes to be associated with intervals rather than single values. In Figure 3.1, an example IDD is shown. It represents the Boolean function $f(u, v, w) = (u \leq 3) \wedge (v \geq 6) \vee (u \geq 4) \wedge (w \leq 7)$ with $u, v, w \in [0, \infty)$.

Equivalent to BDDs, IDDs have a reduced and ordered form, providing a canonical representation of a class of Boolean functions—which is important with respect to efficient fixpoint computations often necessary for formal verification. Methods such as the *If-Then-Else* operator ITE are defined similar to their BDD equivalents and may be computed as usual for decision diagram applications using a computed table to improve performance.

3.2 Interval Mapping Diagrams

IMDs are represented by graphs similar to IDDs. Their edges are labeled with *interval mapping functions* $f : \mathbb{I} \rightarrow \mathbb{I}$ mapping intervals onto intervals, where $\mathbb{I}$

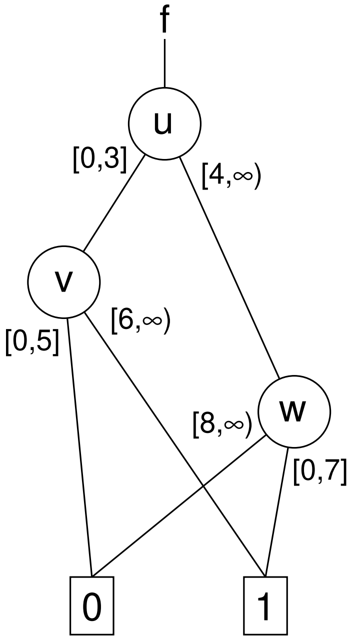

Figure 3.1: Example interval decision diagram.

denotes the set of all integer intervals. The graph contains only one terminal node. In general, IMDs are not canonical. However, this means no restriction for the considered kind of application.

The most important subclass of IMDs are *predicate action diagrams* (PADs) of which the mapping functions are either *shift functions*

$$f_+(I) = \begin{cases} I \cap I_P + I_A & \text{if } I \cap I_P \neq \emptyset \\ [\,] & \text{otherwise} \end{cases}$$

or *assign functions*

$$f_=(I) = \begin{cases} I_A & \text{if } I \cap I_P \neq \emptyset \\ [\,] & \text{otherwise} \end{cases} ,$$

where I_P is the *predicate interval*, I_A the *action interval*, and $+$ stands for interval addition as usual.

The combination of predicate and action interval parameterizes the mapping function and completely defines its behavior. Figure 3.2 shows an example PAD. The syntax $I_P/+I_A$ is used for the shift function f_+ and $I_P/=I_A$ for the assign function $f_=$. The shift about $I = [a, b]$ in reverse direction corresponding to interval substraction is achieved by addition of $-I = [-b, -a] = I_A$ and is denoted as $I_P/-I$.

With regard to transition relations, PADs work as follows. Each edge is labeled with a condition—the predicate interval—on its source node variable and the kind and amount of change—the action operator and the action interval—the variable is to undergo. Each path represents a possible state transition which is executable if all edges along the path are enabled.

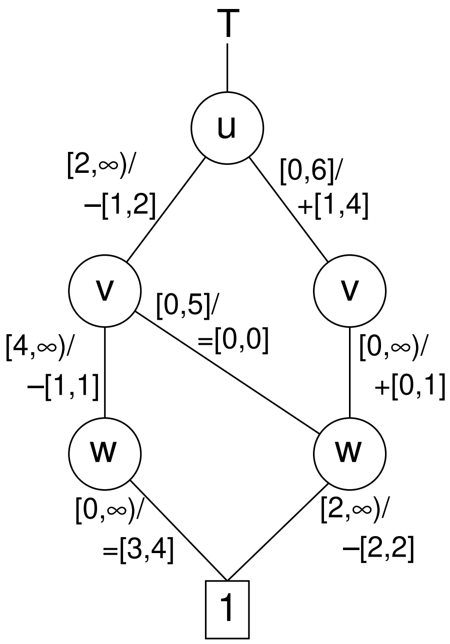

Figure 3.2: Example predicate action diagram.

Chapter 4

Formal Verification of Timed Automata

In this report, we consider only reachability analysis of timed automata. It is performed by iterated application of the transition relation as described in Section 2.2 until reaching a fixpoint. The techniques are directly adaptable for real-time symbolic model checking.

For instance, [HNSY94] considers model checking of timed automata using the real-time logic TCTL over dense time, [ACD93] is similar with regard to *timed graphs*. [CC94] describes symbolic model checking of *timed transition graphs* (TTGs) over discrete time using the logic CTL augmented by a bounded until operator.

Only discrete time represented by integer clock values is considered here. [ABK+97] introduces a discretization scheme transforming dense-time models into discrete-time ones and thus allowing analysis using, e.g., NDDs.

4.1 Using Difference Bounds Matrices

Difference bounds matrices (DBMs) as introduced in [Dil89] may be used for formal analysis of timed automata. DBMs are square matrices of bounds representing convex polyhedra canonically. Unfortunately, non-convex polyhedra, especially unions of convex polyhedra as arbitrary clock regions used in formal verification, have no canonical representations using DBMs, but have to be represented, e.g., by lists of matrices instead. Thus, equivalence testing during fixpoint computation becomes more and more difficult and expensive as the system model grows. Furthermore, DBMs may not easily be combined with symbolic representations of discrete system states.

4.2 Using Numerical Decision Diagrams

Essentially, numerical decision diagrams (NDDs) [ABK+97] are nothing else than BDDs representing sets of integer vectors. The integer elements are coded binarily using "standard positional encoding". The sets to be represented may be described using conjunctions and disjunctions of unequations on integer variables, similar to IDDs. As the binary encoding requires an upper variable value bound, only finite sets may be described in contrast to IDDs. Nevertheless, this is no limitation with respect to formal verification of timed automata.

In contrast to DBMs, NDDs may be used as canonical representations of arbitrary clock regions. [ABK+97] provides a method for formal verification of timed automata using NDDs. While it is based in the main on conventional BDD techniques such as Boolean operations, time projection requires a new algorithm using binary modulo substraction. It is briefly described as a recursive procedure for forward time projection of only one clock, but obviously may be expanded for more than one.

4.3 Using Interval Diagram Techniques

Analogous to above-mentioned models of computation, interval diagram techniques are suitable for formal verification of timed automata due to similar reasons. Discrete-valued clocks may be regarded as particular integer state variables of which the values increase simultaneously when time progresses. Integer time forward projection may be performed by repeated and simultaneous incrementation of all clock values about a *time distance* of 1, depending on the actual system state and thus similar to state distances.

Similar to NDDs, IDDs allow for canonical representations of arbitrary clock regions which is important concerning fixpoint computations. Moreover, they provide a suitable combination with symbolic representations of the discrete part.

Unlike other approaches, our's does not distinguish between time projection and discrete state transitions. Conventionally, both computation stages are performed *alternately*. First, starting with an initial configuration, time is projected to determine all configurations reachable from the inital one by only progressing time. Thereafter, all possible state transitions are performed concurrently, etc. In contrast to this, using interval mapping diagrams allows for a *conjoint* transition behavior consisting of partial time projection—increasing time by one time unit—and discrete state transitions at the same time. This is performed using *image computation* as for conventional reachability analysis. As previous investigations have shown, this seems to be significantly superior to the alternate way with respect to interval diagram techniques. Although more fixpoint iterations are necessary, each one is essentially cheaper than otherwise.

To achieve this, we use a modified transition relation $\tilde{\delta} : Q \times Z \to 2^{Q \times Z}$ with

$$\tilde{\delta}(q, \mathbf{z}) = \Big\{(q', \mathbf{z}') : (\mathbf{z} \in G_{qq} \cap G_{qq'}) \wedge \big(\mathbf{z}' = R_{qq'}(\mathbf{z})\big)\Big\} \cup \Big\{(q, \mathbf{z} + \mathbf{1}) : \mathbf{z} + \mathbf{1} \in G_{qq}\Big\} \tag{4.1}$$

instead of (2.1). This transition relation effectively performs either at most one discrete state transition with respect to the argument configuration or time projection of about exactly one time unit.

We replace (2.2) by using a bounded time forward projection $\widetilde{\Phi} : 2^Z \to 2^Z$ defined as

$$\widetilde{\Phi}(P) = \{\mathbf{z} \in Z : \mathbf{z} \in P \vee \mathbf{z} - \mathbf{1} \in P\}. \tag{4.2}$$

After redefining $\widetilde{P}^\Phi = \widetilde{\Phi}(P) \cap G_{qq}$ and $\widetilde{P}_{q'} = R_{qq'}(P \cap G_{qq} \cap G_{qq'})$ (note the difference to the previous definition here), the immediate successors of a set of configurations (q, P) are denoted—analogous to (2.3)—as

$$\tilde{\delta}\big((q,P)\big) = (q, \widetilde{P}^\Phi) \cup \bigcup_{q' \neq q} (q', \widetilde{P}_{q'}). \tag{4.3}$$

Figure 4.1 shows the transition relation PAD T of the example timed automaton of Figure 2.1. An omitted predicate interval for the mapping functions means "no condition" for the respective variable, while an omitted action results in no variable value change. Single integer values stand for singleton intervals with this only element.

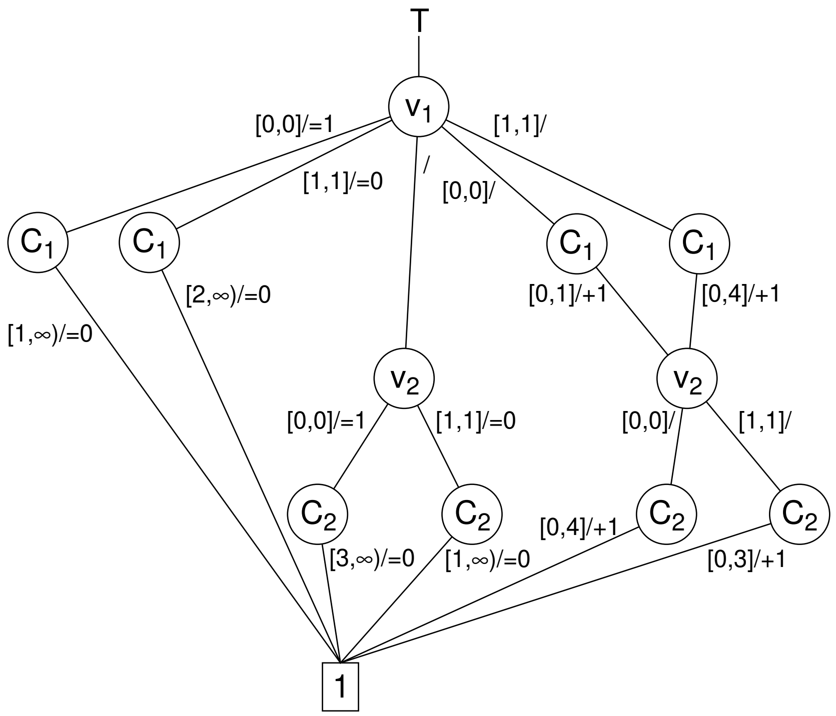

Figure 4.1: Transition relation PAD.

The two left-most paths of the PAD T describe the transition guards, state changes, and reset functions resulting from both transitions, respectively, of the upper partial automaton. For instance, the top-most transition is enabled if $(v_1 = 0) \wedge (C_1 \geq 1)$, i.e., $v_1 \in [0,0] \wedge C_1 \in [1,\infty)$. The consequence of this transition is that state variable v_1 is set to 1, and clock C_1 is reset to 0. Similarly, the two paths in the middle of T represent the transitions of the lower partial automaton.

Altogether, the paths of both automata describe the right argument of the union operator in (4.3).

The right-most paths—four altogether—are required to model time progress depending on the actual state. Time can only progress if all clock values are increased simultaneously by not violating any of the staying conditions. The clocks increase about one time unit per step, but only if the respective conditions depending on the system state are satisfied. Thus, these paths describe the left argument of the union operator in (4.3) except for (q, P) which is added algorithmically to the final result for $\tilde{\delta}\big((q, P)\big)$ later on.

In [ST98b], an efficient algorithm is described to perform image computation using an IDD S for the state set and a PAD T for the transition relation, resulting in an IDD S' representing the image state set. This algorithm may be used to perform reachability analysis or real-time symbolic model checking by fixpoint computation.

Chapter 5

Empirical Results

In [ABK+97], two parameterized models are used to compare the NDD and the DBM approach. As NDDs seem to be greatly superior to DBMs—which on the other hand are suitable for directly handling dense time—regarding computation time and memory resources, only NDDs are considered here. We compare their runtime behavior to that of the interval techniques approach.

The examples used are a timed automaton $\mathcal{A}$ with one discrete state and an automaton $\mathcal{B}$ with many states—shown in Figure 5.1 a) and Figure 5.1 b), respectively. A configuration parameter n indicates the number of self-loop transitions from and to $\mathcal{A}$'s only location or the number of concurrent partial automata—each consisting of two locations and two transitions—of $\mathcal{B}$, respectively. For both $\mathcal{A}$ and $\mathcal{B}$, n denotes the number of clocks as well. The total number of states of $\mathcal{B}$ is 2^n.

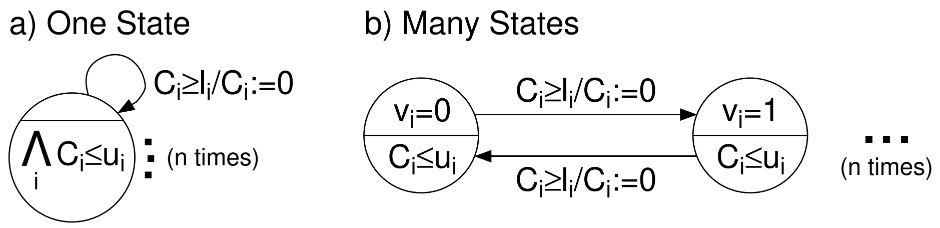

Figure 5.1: Timed automata $\mathcal{A}$ with one state and $\mathcal{B}$ with many states.

$\mathcal{A}$ may be used to model a system generating n events $\tau_1, ..., \tau_n$ such that every occurrence of τ_i must be followed by another one within u_i time units, while every two occurrences of τ_i must be separated by l_i units. $\mathcal{B}$ may represent n boolean input signals of which the only constraints are that every two changes in their values are separated by some time between l_i and u_i. Such automata are indispensable for analyzing system behaviors under all possible inputs.

As no NDD implementation was available and [ABK+97] and [BMPY97] do only sketch the used algorithms, the comparison had to be performed based on the results reported in [ABK+97], obtained on a Sun Ultra 1 with 170 MHz. The results considering computation time have been downscaled due to different computing powers—for our experiments, we used a Sun Ultra 30 with 300 MHz—using a factor of 1.7, determined by comparative experiments on both machines.

In Figure 5.2, the computation time T to determine the set of reachable configurations of the "one state" automaton $\mathcal{A}$ is depicted in logarithmic scale, depending on the configuration parameter n.

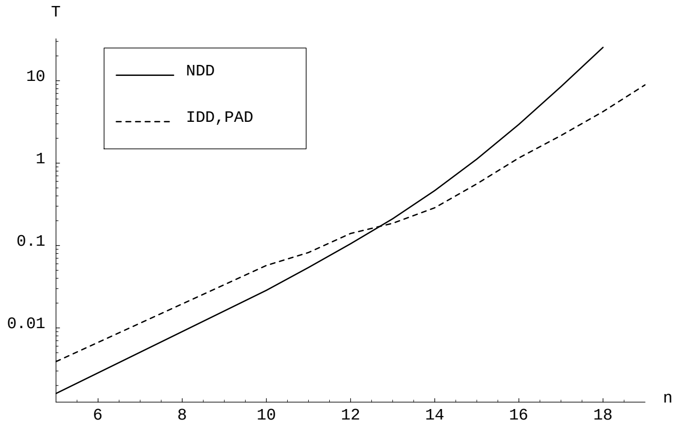

Figure 5.2: Computation time T in 10^3 seconds for "one state" timed automaton $\mathcal{A}$.

The "many states" example behaves very similar. Figure 5.3 shows the computation time of reachability analysis for automaton $\mathcal{B}$.

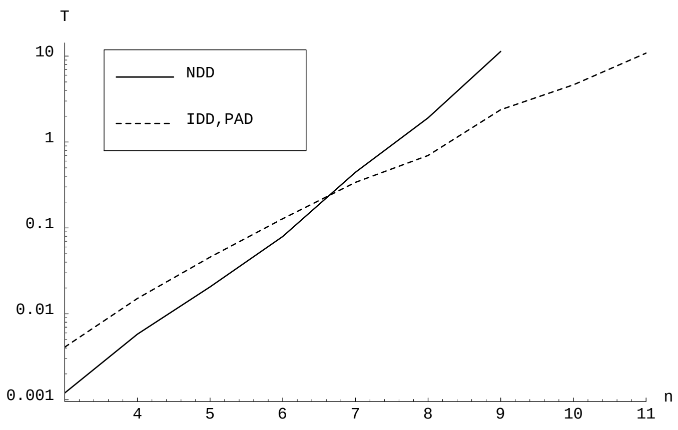

Figure 5.3: Computation time T in 10^3 seconds for "many states" timed automaton $\mathcal{B}$.

For large models, the IDD/PAD approach significantly outperforms the NDD approach. The break-even occurs in the region of several minutes of computation time which is of importance especially with regard to practical application. Most noteworthy is that the weaker gradient of the IDD/PAD computation time for both examples seems to be an indication that the algorithmic complexity for this kind of application is better.

For our implementation, we used the programming language Java for experimental purposes, while the NDD approach was implemented in C. Current Java compilers and interpreters achieve implementation speeds which are about 5 to 10 times lower than those of C. Hence, equivalent implementations should shift the IDD/PAD graph about up to one decade downward such that our approach outperforms the NDD approach even for small parameter values of n.

Chapter 6

Summary and Conclusion

An approach for formal verification of timed automata using interval diagram techniques has been presented. IDDs and IMDs have been explained together with the verification methods necessary for reachability analysis and real-time symbolic model checking of timed automata. Our results outperform those of the NDD approach with regard to computation time.

Without further modifications, interval diagram techniques may be applied to the analysis of discrete hybrid automata on integer variables—analogous to timed automata—by replacing the common clock addition of $+[1, 1]$ by non-uniform integer rates.

Bibliography

[ABK+97] E. Asarin, M. Bozga, A. Kerbrat, O. Maler, A. Pnueli, and A. Rasse. Data-structures for the verification of timed automata. In O. Maler, editor, *Hybrid and Real-Time Systems*, volume 1201 of *Lecture Notes in Computer Science*. Springer-Verlag, 1997.

[ACD93] R. Alur, C. Courcoubetis, and D. Dill. Model checking in dense real-time. *Information and Computation*, 104(1):2–34, May 1993.

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

[BMPY97] M. Bozga, O. Maler, A. Pnueli, and S. Yovine. Some progress in the symbolic verification of timed automata. In *Proceedings of the 9th International Conference on Computer-Aided Verification*, volume 1254 of *Lecture Notes in Computer Science*. Springer-Verlag, 1997.

[Bry86] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):667–691, August 1986.

[CC94] S. V. Campos and E. M. Clarke. *Theories and Experiences for Real-Time System Development*, chapter Real-time symbolic model checking for discrete time models. AMAST Series in Computing. World Scientific, 1994.

[Dil89] D. L. Dill. Timing assumptions and verification of finite-state concurrent systems. In J. Sifakis, editor, *Automatic Verification Methods for Finite State Systems*, volume 407 of *Lecture Notes in Computer Science*. Springer-Verlag, 1989.

[HNSY94] T. A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111(2):193–244, June 1994.

[MP95] O. Maler and A. Pnueli. Timing analysis of asynchronous circuits using timed automata. In P. E. Camurati and H. Eveking, editors, *Correct Hardware Design and Verification Methods*, volume 987 of *Lecture Notes in Computer Science*. Springer-Verlag, 1995.

[SKMB90] A. Srinivasan, T. Kam, S. Malik, and R. K. Brayton. Algorithms for discrete function manipulation. In *Proceedings of the IEEE International Conference on Computer Aided Design*, 1990.

[ST98a] Karsten Strehl and Lothar Thiele. Symbolic model checking of process networks using interval diagram techniques. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD-98)*, 1998.

[ST98b] Karsten Strehl and Lothar Thiele. Symbolic model checking using interval diagram techniques. Technical Report 40, Computer Engineering and Networks Lab (TIK), Swiss Federal Institute of Technology (ETH) Zurich, Gloriastrasse 35, CH-8092 Zurich, February 1998.