In *Proceedings of the 6th International Conference on Real-Time Computing Systems and Applications (RTCSA '99)*,
Hong Kong, pages 488–491, December 13–15, 1999.

Interval Diagrams: Increasing Efficiency of Symbolic Real-Time Verification

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/~symbolic

Abstract

In this paper, we suggest *interval diagram techniques* for formal verification of real-time systems modeled by means 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. Our approach is used for performing both timed reachability analysis and real-time symbolic model checking. We present the methods necessary for our approach and compare its results to another, similar verification technique—achieving a speedup of 7 and more.

1 Introduction

Especially for safety-critical real-time applications such as those in traffic control, medical engineering, or avionics, simulation often is not sufficient to guarantee the correctness of a technical system's model. In addition to simulation, formal methods are employed to verifiy the system behavior and to determine timing properties. Many approaches have been introduced to modeling timing behavior of real-time systems, mostly derived from conventional finite state automata which are expanded to describe timing properties of the transition behavior. As one of the most universal approaches, Alur and Dill have proposed *timed automata* [1], represented by state-transition graphs with timing constraints using finitely many clocks.

There exist several classes of techniques to perform formal verification of real-time systems. Timed reachability analysis determines the set of states reachable from a given set of initial states. The meaning of the term *state* here comprises in addition to the usual discrete system state some actual timing information of the system describing, e.g., part of the timed execution history. Reachability analysis allows for proving that, e.g., an error state may never be reached or certain timing constraints are always satisfied. 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. Complex properties such as the guaranteed acknowledgement response within a specified amount of time after submitting a request can be proven.

Reachability analysis of timed automata often is performed using *difference bounds matrices* (DBMs) [5] for representing *clock regions* during computation. As DBM methods often fail for large models, other approaches have been proposed using different kinds of region representations. Besides methods extending and improving DBM approaches by using, e.g., more compact data structures or on-the-fly reduction, DBM independent methods such as those based on *binary decision diagrams* (BDDs) [6], *numerical decision diagrams* (NDDs) [2, 4], and *multi-terminal BDDs* (MTBDDs) [7] (the latter both are derivatives of BDDs) 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 [9] or Petri nets [10], often providing advantages with regard to computation time and memory resources. Furthermore, applications in the area of scheduling of heterogeneous embedded systems are possible [11]. In this paper, interval diagram techniques are applied to formal verification of timed automata in a multi-clock setting over discrete time. We present the used interval diagrams and verification techniques and compare their run-time behavior with that of the NDD approach. Due to the lack of space, we have to refer to [8] for more detailed explanations.

2 Timed Automata

Figure 1 shows an example timed automaton. It may be used for modeling two independent non-determinate input oscillators (denoted by v_1 and v_2) of which the pulse widths are known only within certain time ranges. The length of the 0-pulse of v_1 is in the range of $[1,2]$, while the 1-pulse's length lies in $[2,5]$. For v_2, the ranges are $[3,5]$ and $[1,4]$, respectively.

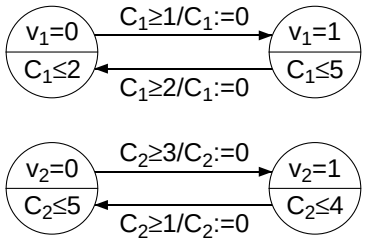

Figure 1: Example timed automaton.

First, we give a brief and informal introduction to timed automata. They will be defined formally later on. The automaton of Figure 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. v_1 and v_2 are state variables encoding the discrete automaton states. The product of the locations of the partial automata results in four discrete states $q \in Q$ with $q = (v_1, v_2)$ and $Q = \{(0,0),(0,1),(1,0),(1,1)\}$. 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 satisfied 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 [1]. In this section, we use the following definitions analogous to [2]. Bold-

face letters are used for denoting 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 $\mathbf{u},\mathbf{v} \in \mathbb{R}^d$, $\mathbf{u} \le \mathbf{v}$ denotes that $u_i \le v_i$ 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 $\mathbf{u} \le \mathbf{v}$, $\mathbf{v} \in S$ 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 $\{\mathbf{v} : v_i \le c\}$, $\{\mathbf{v} : v_i < c\}$, $\{\mathbf{v} : v_i - v_j \le c\}$, 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* [1] 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. 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 (Timed Automaton) *A timed automaton is a triple $A = (Q,Z,\delta)$ 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 $(q,\mathbf{z}) \in Q \times Z$,*

$$\delta(q,\mathbf{z}) = \Big\{ (q',\mathbf{z}') : \exists t \in K \text{ such that } (\mathbf{z}+t \in G_{qq} \cap G_{qq'}) \wedge \big(\mathbf{z}' = R_{qq'}(\mathbf{z}+t)\big) \Big\}. \quad (1)$$

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 is defined as a function $\Phi : 2^Z \to 2^Z$ with

$$\Phi(P) = \{\mathbf{z}+t : \mathbf{z} \in P, t \in K\} \cap Z. \quad (2)$$

(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

$$\delta\big((q,P)\big) = (q,P^\Phi) \cup \bigcup_{q'\neq q} (q',P_{q'}). \quad (3)$$

3 Interval Diagram Techniques

Using interval diagram techniques—based on interval decision diagrams (IDDs) and interval mapping diagrams (IMDs)—for formal verification of process networks [9] and Petri nets [10] reduced the computation times by a factor of up to more than 5 compared to BDD techniques.

IDDs are a generalization of BDDs and MDDs—*multi-valued decision diagrams*—allowing diagram variables to be integers and child nodes to be associated with intervals rather than single values. Analogously 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. IDDs are used for representing state sets.

IMDs are represented by graphs similar to IDDs. Their edges are labeled with *interval mapping functions* $f : \mathbb{I} \to \mathbb{I}$ mapping intervals onto intervals, where $\mathbb{I}$ denotes the set of all integer intervals. The graph contains only one terminal node. In the scope of this paper, 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. Figure 2 shows an example IMD. Its relationship to a timed automaton is explained later on. The syntax $I_P/+I_A$ is used for the shift function f_+ and $I_P/=I_A$ for the assign function $f_=$.

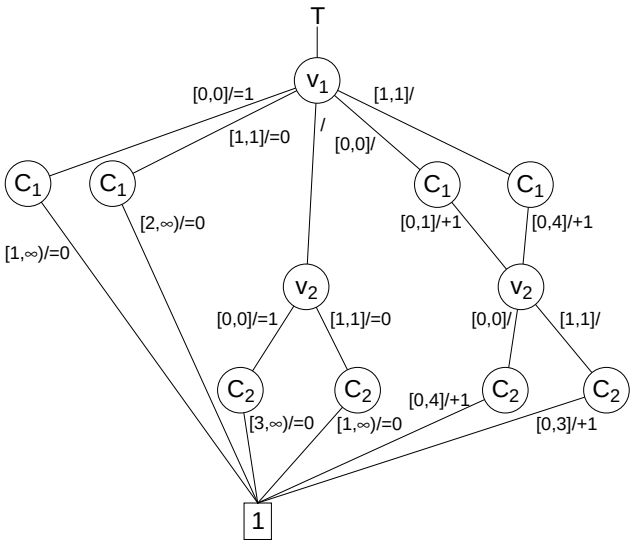

Figure 2: Transition relation IMD.

With regard to transition relations, IMDs 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.

4 Formal Verification of Timed Automata

In the scope of this paper, mainly reachability analysis of timed automata in a multi-clock setting is considered. It is performed by iterated application of the transition relation δ as described in Section 2.2 until reaching a fixpoint.

Besides the application of the transition relation, equality tests of sets of configurations are crucial operations in reachability analysis with regard to efficiency. Hence, having a canonical configuration set representation is a fundamental factor because testing for equality then often requires only constant time.

Only discrete time represented by integer clock values is considered here. [2] introduces a discretization scheme transforming dense-time models into discrete-time models and thus allowing analysis using NDDs or interval diagram techniques.

4.1 Using Difference Bounds Matrices

Difference bounds matrices (DBMs) as introduced in [5] 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, equality 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) [2] are BDDs representing sets of integer vectors. The integer elements are coded binarily. The sets to be represented may be described using conjunctions and disjunctions of unequations on integer variables, similarly to IDDs. As the binary encoding requires an upper variable value bound, only finite sets may be described in contrast to IDDs. In contrast to DBMs, NDDs may be used as canonical representations of arbitrary clock regions. [2] provides a method for discrete-time formal verification of timed automata using NDDs.

4.3 Using Interval Diagram Techniques

Analogously 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 can be performed by repeated and simultaneous incrementation of all clock values about a *time distance* of 1, depending on the actual system state.

Similarly to NDDs, IDDs allow for canonical representations of arbitrary clock regions which is important with regard to fixpoint computations as mentioned above. Moreover, they provide a suitable combination with symbolic representations of the discrete part of the model.

Unlike other approaches, ours does not distinguish between time projection and discrete state transitions. Conventionally, both computation stages are performed *alternately*. 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.

We use a modified transition relation $\tilde{\delta} : Q \times Z \to 2^{Q \times Z}$ with

$$\tilde{\delta}(q,\mathbf{z}) = \Big\{ (q',\mathbf{z}') : (\mathbf{z} \in G_{qq} \cap G_{qq'}) \wedge \big(\mathbf{z}' = R_{qq'}(\mathbf{z})\big) \Big\} \cup \Big\{ (q,\mathbf{z}+\mathbf{1}) : \mathbf{z}+\mathbf{1} \in G_{qq} \Big\} \quad (4)$$

instead of (1). This transition relation effectively performs either at most one discrete state transition with respect to the argument configuration or time projection of exactly one time unit.

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

$$\widetilde{\Phi}(P) = \{\mathbf{z} \in Z : \mathbf{z} \in P \vee \mathbf{z}-\mathbf{1} \in P\}. \quad (5)$$

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—analogously to (3)—as

$$\tilde{\delta}\big((q,P)\big) = (q,\widetilde{P}^{\Phi}) \cup \bigcup_{q' \neq q} (q',\widetilde{P}_{q'}). \quad (6)$$

Figure 2 shows the transition relation IMD T of the example timed automaton of Figure 1. The two left-most paths of the IMD 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 $(v_1 = 0) \wedge (C_1 \geq 1)$ is satisfied, 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 (6).

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 while 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 (6) except for (q,P) which is added algorithmically to the final result for $\tilde{\delta}\big((q,P)\big)$ later on.

In [9], an efficient algorithm is described to perform image computation using an IDD S for the state set and an IMD T for the transition relation, resulting in an IDD S' representing the image state set. This algorithm may be used for performing reachability analysis or real-time symbolic model checking by fixpoint computation.

4.4 Real-Time Symbolic Model Checking

Besides reachability analysis, the techniques described above are directly applicable to real-time symbolic model checking. Especially the approach of Ruf and Kropf [7] to real-time symbolic model checking of *interval structures* using *clocked* CTL (CCTL)—a derivative of the well-known branching-time temporal logic *computation tree logic* (CTL)—over discrete time is well-suited for applying interval diagram techniques.

The CCTL operator $\mathsf{EX}_{[1]}$, of which other temporal operators are composed, is computed using an IMD similar to that of Figure 2 representing the timed transition relation. The necessary pre-image operation then is performed using the techniques for backward image computation as described in [9], replacing the above-mentioned forward image computation.

5 Empirical Results

In [2], two parameterized models are used for comparing the NDD and the DBM approach. In addition to DBM methods, the NDD techniques have been implemented in the real-time formal verification tool *Kronos*. As NDDs seem to be greatly superior to DBMs with regard to computation time and memory resources, only NDDs

are considered here. We compare their run-time behavior to that of the interval diagram techniques approach.

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

In Figure 3, the computation time T to determine the set of reachable configurations of the "one state" automaton A is compared for the NDD and the IDD/IMD approach, depending on the configuration parameter n. T is given in seconds and depicted in logarithmic scale. Details about the experimental setting are described in [8].

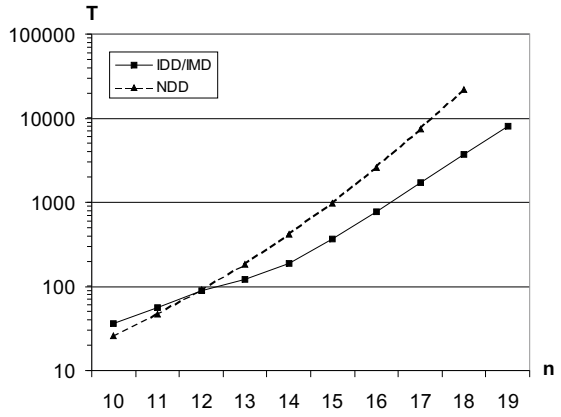

Figure 3: Computation time T for "one state" automaton A.

The "many states" automaton B behaves very similarly. Figure 4 shows the computation time of its reachability analysis. Note that again a logarithmic scale is chosen for the sake of comparability, meaning that even small differences in the plot may correspond to large differences in the computational speed.

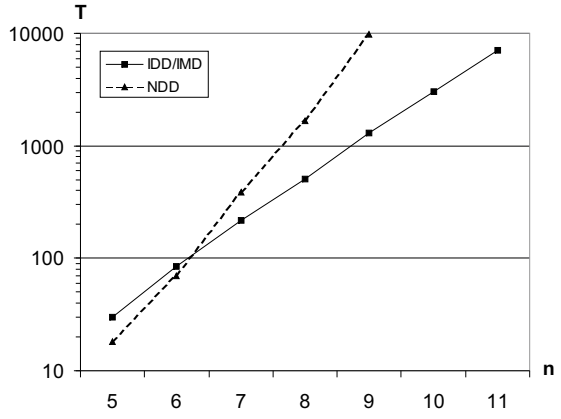

Figure 4: Computation time T for "many states" automaton B.

For medium and large models, the IDD/IMD approach significantly outperforms the NDD approach. Most noteworthy is that the lesser slope of the IDD/IMD computation time seems to be an indication that the algorithmic complexity for this kind of application is better. The highest speedup achieved is more than 7. It is expected to be even greater for larger models—of which no NDD computation times have been available for comparison. Further, we compared our experimental implementation with the highly efficient state-of-the-art tool *Kronos*.

6 Concluding Remarks

6.1 Further Related Work

Besides the NDD approach described in Section 4.2, especially two other related approaches are relevant. In [6], vectors of BDDs are used as a representation of the transition relation. This representation has been replaced in [7] for efficiency reasons by using MTBDDs as state set representation. Meanwhile, the MTBDD approach yielded a speedup of up to 3.5 compared to *Kronos*.

An approach inspired by interval diagram techniques is introduced in [3]. It makes use of *clock difference diagrams* (CDDs) which are based on a modification of IDDs but unfortunately abandon a directly canonical representation. For equality tests during fixpoint computations, CDDs first have to be transformed into a quasi-canonical form. This seems to limit run-time reductions significantly which is confirmed by the experimental results presented—even slight run-time increases compared to DBMs.

6.2 Summary

An approach to formal verification of real-time systems modeled by means of timed automata has been presented which makes use of interval diagram techniques. IDDs and IMDs have been presented together with the verification methods necessary for timed analysis. Reachability analysis in a multi-clock setting over discrete time has been considered, and extensions for real-time symbolic model checking have been sketched. Our results outperform those of the NDD approach with regard to computation time by a factor of more than 7 and likely even more.

References

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

[2] 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.

[3] G. Behrmann, K. G. Larsen, J. Pearson, C. Weise, and W. Yi. Efficient timed reachability analysis using clock difference diagrams. In *Computer Aided Verification*, volume 1633 of *Lecture Notes in Computer Science*, pages 341–353. Springer-Verlag, 1999.

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

[5] 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.

[6] J. Frößl, J. Gerlach, and T. Kropf. An efficient algorithm for real-time model checking. In *Proceedings of the European Design and Test Conference*, pages 15–21, 1996.

[7] J. Ruf and T. Kropf. *Advances in Hardware Design and Verification*, chapter Symbolic Model Checking for a Discrete Clocked Temporal Logic with Intervals, pages 146–163. Chapman & Hall, 1997.

[8] K. Strehl. Using interval diagram techniques for the symbolic verification of timed automata. Technical Report TIK-53, Computer Engineering and Networks Lab (TIK), Swiss Federal Institute of Technology (ETH) Zurich, Gloriastrasse 35, CH-8092 Zurich, July 1998.

[9] K. Strehl and L. 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)*, pages 686–692, San Jose, California, November 8–12, 1998.

[10] K. Strehl and L. Thiele. Interval diagram techniques for symbolic model checking of Petri nets. In *Proceedings of the Design, Automation and Test in Europe Conference (DATE99)*, pages 756–757, Munich, Germany, March 9–12, 1999.

[11] K. Strehl, L. Thiele, D. Ziegenbein, R. Ernst, and J. Teich. Scheduling hardware/software systems using symbolic techniques. In *Proceedings of the 7th International Workshop on Hardware/Software Codesign (CODES'99)*, pages 173–177, Rome, Italy, May 3–5, 1999.