Space Efficient Reachability Analysis for a Value Passing, Timed Process Algebra
(Extended Abstract)

David Kendall, William Henderson, and Adrian Robson

Department of Computing and Mathematics, University of Northumbria at Newcastle, Ellison Building, Newcastle upon Tyne, NE1 8ST
{david.kendall,william.henderson,adrian.robson}@unn.ac.uk

Abstract. Reachability analysis and model checking of timed automata are now well-established techniques in the analysis of real-time control systems. The major limiting factor in their use, from a technical point of view, remains the state explosion problem. Symbolic representation of the state space often allows for the analysis of much larger systems than the point-wise representation which is common in enumerative analysis. In particular, the use of binary decision diagrams (BDD's) has been successful, mainly in the analysis of hardware systems where the need for a compact representation of boolean functions is prevalent. However, the modelling of software systems commonly employs a richer set of data types and this fact motivates the investigation of different encodings of sets of states than by their characteristic functions. This paper considers the use of minimized deterministic finite state automata (MA's) for the storage of the set of visited states in the reachability analysis of models of broadcasting systems expressed using a timed process algebra.

1 Introduction

This paper outlines an approach to the implementation of reachability analysis of system models expressed using *bCANDLE* [16], an algebra of asynchronous broadcasting systems, which we have developed to facilitate the modelling of distributed control systems which are implemented using Controller Area Network (CAN) [15]. Although we are interested in a rather specialized class of systems, the results in this paper are significant in the wider context of the reachability analysis of timed automata [2].

The rest of the paper is organised as follows: Sect. 2 briefly introduces *bCANDLE*; Sect. 3 outlines an approach to timed reachability analysis of *bCANDLE* models and introduces the main data structures needed to support the analysis; MA's are introduced in section 4 and their application in reachability analysis is considered; our implementation and experimental results are described in Sect. 5; related work is reviewed in Sect. 6; Sect. 7 concludes and proposes further work.

2 *bCANDLE*

bCANDLE is a timed process algebra allowing the expression of system models consisting of a set of asynchronous processes, each having its own local data state and communicating by sending and receiving messages on one or more *broadcast* channels which implement the CAN protocol.

A process can engage in three different kinds of basic action:

1. sending a message on a channel;
2. receiving a message from a channel, and
3. performing an operation in a bounded amount of time, possibly transforming its local data state on completion.

More complex process behaviour can be defined using sequential composition, guards on data states, non-deterministic choice and interrupt. A *bCANDLE* system consists of the asynchronous parallel composition of a fixed number of processes together with a network of CAN channels.

The semantics of a *bCANDLE* system is given by associating with it a timed transition system. A *timed transition system* $\mathcal{S} = (\Sigma, \sigma^{\mathcal{I}}, \mathcal{L}, \longrightarrow)$ is a tuple in which Σ is the set of states, $\sigma^{\mathcal{I}} \in \Sigma$ is the initial state, $\mathcal{L} = \mathrm{A} \cup \mathbb{R}^{>0}$ is the set of labels consisting of *action* labels A and *time* labels $\mathbb{R}^{>0}$ and $\longrightarrow \subseteq \Sigma \times \mathcal{L} \times \Sigma$ is the transition relation which is required to be deterministic and continuous with respect to the passage of time. A detailed semantics of *bCANDLE* is given in [16].

For the purposes of verifying a *bCANDLE* system, we work with a timed safety automaton [13] having a bisimilar transition system. We introduce timed safety automata briefly below.

2.1 Timed Safety Automata

Let H be a set $\{h_0, h_1, \cdots, h_n\}$ of real-valued variables, called *clocks*, where each $h_i \in \mathrm{H}$ ranges over the non-negative reals $\mathbb{R}^+$. A *clock valuation* is a function $\mathbf{v} : \mathrm{H} \to \mathbb{R}^+$ which assigns a value in $\mathbb{R}^+$ to each clock in H. We assume that h_0 is given the value 0 by every clock valuation. We denote by $\mathbf{v}[\mathrm{H}' := 0]$, the clock valuation $\mathbf{v}'$ such that $\mathbf{v}'(h) = 0$ for all $h \in \mathrm{H}'$ and $\mathbf{v}'(h) = \mathbf{v}(h)$ otherwise. For $t \in \mathbb{R}^+$, we denote by $\mathbf{v} + t$ the clock valuation $\mathbf{v}'$ such that $\mathbf{v}'(h) = \mathbf{v}(h) + t$ for all clocks in H. $\mathbf{0}$ is the clock valuation which assigns 0 to every clock. A *bound* over H is a constraint of the form $h_i - h_j \# c$ where $i, j \in \{0, \ldots, n\}$, $\# \in \{<, \leq\}$ and $c \in \mathbb{Z} \cup \infty$. A *clock constraint* is a conjunction of bounds; it defines a convex subset of $\mathbb{R}^n$ known as a *zone*. We denote the set of clock constraints over the clocks H by $\Phi(\mathrm{H})$.

A *timed safety automaton* (TSA) is a tuple $\mathcal{A} = (\mathrm{Q}, q^{\mathcal{I}}, \mathrm{A}, \mathrm{E}, \mathrm{H}, \mathrm{I})$ where: Q is a finite set of *control locations*, $q^{\mathcal{I}}$ is the initial control location, A is a finite set of event names, E is a finite set of edges where an edge is of the form (q, ϕ, a, X, q') where $q, q' \in \mathrm{Q}$ are control locations; $\phi \in \Phi(\mathrm{H})$ is a clock constraint, $a \in \mathrm{A}$ is an event name and $X \subseteq \mathrm{H}$ is a set of clocks to be reset; H is a finite set of clocks,

and $\mathrm{I} : \mathrm{Q} \rightarrow \Phi(\mathrm{H})$ is a function which associates a *time progress* condition (or *invariant*) with each control location.

The semantics of the TSA $\mathcal{A} = (\mathrm{Q}, q^{\mathcal{I}}, \mathrm{A}, \mathrm{E}, \mathrm{H}, \mathrm{I})$ is given by the timed transition system $\mathcal{T}[\![A]\!] = (\Sigma, \sigma^{\mathcal{I}}, \mathcal{L}, \longrightarrow)$ where Σ is the set of pairs $(q, \mathbf{v})$ such that $q \in \mathrm{Q}$ is a location of A and $\mathbf{v}$ is a clock valuation for H which satisfies the invariant $\mathrm{I}(q)$; $\sigma^{\mathcal{I}} = (q^{\mathcal{I}}, \mathbf{0})$ is the initial state; $\mathcal{L} = \mathrm{A} \cup \mathbb{R}^{>0}$ is the set of labels; $\longrightarrow \subseteq \Sigma \times \mathcal{L} \times \Sigma$ is the transition relation containing transitions as follows:

- *Time transitions*: A state can change due to the elapse of time. There is a transition $(q, \mathbf{v}) \xrightarrow{t} (q, \mathbf{v} + t)$ if for all $t' \leq t$, $\mathbf{v} + t'$ satisfies the invariant $\mathrm{I}(q)$.
- *Event transitions:* A state can change by moving location. For each state $(q, \mathbf{v}) \in \Sigma$, if there is an edge $(q, \phi, a, X, q') \in \mathrm{E}$ such that $\mathbf{v}$ satisfies ϕ, then there is a transition $(q, \mathbf{v}) \xrightarrow{a} (q', \mathbf{v}[X := 0])$.

2.2 From *bCANDLE* to Timed Safety Automata

The translation of the timed algebra ATP [19] into TSA is shown in [24]. We have adapted this approach in order to associate a TSA with a *bCANDLE* system. The main addition is to handle the additional *context* i.e. the network and data environment. In this approach the control part of the system model is first translated into a net-like structure. In any state of the system, the currently active transitions are given by a marking of the net. Each transition has a number of *attributes* associated with it. The attributes of a transition might include a context condition, a label, a clock and upper and lower bounds for the clock value. If the context and clock conditions are satisfied, a marking M and a context C may be transformed by an active transition to a new marking M' and a new context C', possibly resetting some clocks along the way. Lack of space prevents us from giving the details here, however, from this brief description it can be seen that for a TSA derived from a *bCANDLE* system model, a location comprises information about: 1) the current marking, given as a set of integers $\{p_1, p_2, \ldots, p_c\}$ for some variable number c where each p_i identifies an active transition; 2) the values of all data variables, given as a sequence of values $\langle v_1, v_2, \ldots, v_d \rangle$ for some fixed number d of variables; and 3) the state of the network channels, given as a sequence of pairs (s, m) where s gives the status of the channel, *free, transmitting*, etc. and m is a variable-length, priority ordered sequence of messages awaiting transmission on the channel.

A state in a TSA is given by a location and a clock valuation. However, we will see in Sect. 3 that a practical implementation of timed reachability analysis requires the use of *symbolic states* where each symbolic state represents a location and a *set* of clock valuations.

A *bCANDLE* state vector, then, consists of a location as described above and a zone defining a set of clock valuation(s), see Fig. 1.

$\{p_1, p_2, \cdots, p_c\}$	$\langle v_1, v_2, \cdots, v_d \rangle$	$\langle (s_1, m_1), (s_2, m_2), \cdots, (s_n, m_n) \rangle$	$\langle \phi_1, \phi_2, \cdots, \phi_r \rangle$
Control	Data	Network	Clock Valuations
LOCATION – q			ZONE – Z

Fig. 1. Structure of a *bCANDLE* state vector

3 Reachability Analysis

Of the several approaches to automated analysis of real-time systems, reachability analysis (RA) is one of the more easily implemented and informative. The basic reachability problem is to determine the set of system states which are encountered on any execution starting from some given state. RA allows the checking of *safety* properties of a system by answering the question: is it possible to reach an incorrect or unsafe state from an initial state. Other verification problems can be solved by building upon the solution of the basic RA problem [1]. The algorithm of Fig. 2 shows how to compute the set of states reachable from a given initial state.[1]

```
VISITED := {(q^I, 0)}
WAITING := {(q^I, 0)}
while WAITING ≠ ∅ do
  remove some (q, Z) from WAITING
  succ := {(q_s, Z_s) | (q, Z) ⟶_S (q_s, Z_s) ∧ Z_s ≠ ∅}
  foreach (q_s, Z_s) ∈ succ do
    if (q_s, Z_s) ∉ VISITED
      add (q_s, Z_s) to VISITED
      add (q_s, Z_s) to WAITING
    fi
  od
od
```

Fig. 2. An algorithm for reachable states

The termination of the algorithm depends upon the construction of a finite quotient of the infinite transition system given by the TSA semantics. Important aspects of such a construction are 1) the use of *symbolic states* (q, Z) where q is a location as usual and Z is a zone which represents a set of clock valuations, and 2) the definition of a transition relation $\longrightarrow_{\mathrm{S}}$ between symbolic states. Essentially, for a symbolic state $S = (q, \mathrm{Z})$ and an edge $e = (q, \mathrm{Z}_e, a, X, q')$, we

[1] Both KRONOS and UPPAAL implement a more efficient algorithm using inclusion abstraction in which the test $(q_s, \mathrm{Z}_s) \notin \mathit{VISITED}$ is replaced by the more efficient test $\forall (q_s, \mathrm{Z}') \in \mathit{VISITED} \bullet \mathrm{Z}_s \not\subseteq \mathrm{Z}'$. We will implement this improvement soon.

say $S \longrightarrow_{\mathrm{S}} \mathbf{post}_e(S)$ where

$$\mathbf{post}_e(q, \mathrm{Z}) = (q', \mathrm{I}(q') \cap \nearrow((\mathrm{Z} \cap \mathrm{Z}_e)[X := 0])$$

Here $\nearrow \mathrm{Z}$ is the time progress operator which relaxes all upper bounds and $\mathrm{Z}[X := 0]$ is the clock reset operator. See [8] for further details.

The problem considered in the rest of the paper is how to arrange for the compact storage of the set *VISITED* of visited state vectors, where a major difficulty has been to combine a good encoding of the locations (the discrete part of the system) with a good encoding for the clock valuations (the continuous part). This problem is addressed in the following section.

4 A Minimized Automaton Representation of Reachable States

A state vector can be regarded as being encoded as a string of bytes. This opens up the possibility of representing the set of visited states by a deterministic finite automaton (DFA) which recognizes it. Such an approach has been implemented in the model checker SPIN [14] where it has been shown to achieve even better compression for many systems than that obtained by the use of BDD's [22]. The approach has not been previously applied in the analysis of timed systems.

Definition 1. *A k-layer DFA is* $A = (Q, \delta, \Sigma)$ *where* $Q = \bigcup\{Q_i \mid 0 \leq i \leq k\}$ *is the set of states,* $Q_i \subset Q$ *is the non-empty set of states at the ith layer and* $Q_i \cap Q_j = \emptyset$ *for* $i \neq j$. Q_0 *is a singleton containing the initial state and* $Q_k = \{\mathrm{T}, \mathrm{F}\}$. Σ *is the alphabet and* $\delta : Q^- \times \Sigma \to Q$ *is the transition function where* $Q^- = Q \setminus Q_k$ *and for all states* $q \in Q^-$ *and symbols* $\sigma \in \Sigma$, *if* $q \in Q_i$ *then* $\delta(q, \sigma) \in Q_{i+1}$. T *is the accepting final state and* F *is the rejecting final state.*

We use $\sigma[i, j]$ to denote the string $\sigma_i, \sigma_{i+1}, \ldots, \sigma_j$. A string $\sigma[i, j-1] \in \Sigma^{j-i}$ generates a run $q_i q_{i+1}, \ldots, q_j$ where $\delta(q_m, \sigma_m) = q_{m+1}$. We define $\mathcal{L}_A(q_i) = \{\sigma[i, k-1] \mid \sigma[i, k-1]$ generates the run $q[i, k]$ where $q_k = \mathrm{T}\}$. We define $\mathcal{L}(A) = \mathcal{L}_A(q_0)$ where $q_0 \in Q_0$. A DFA is *minimized* provided $\mathcal{L}(q_i) = \mathcal{L}(q_j)$ iff $q_i = q_j$.

Example 1. The MA of Fig. 3 is $A = (\{Q_i\}_{i=0}^4, \delta, \Sigma)$ where $Q_0 = \{0\}, Q_1 = \{1, 2, 3\}, Q_2 = \{4, 5\}, Q_3 = \{6, 7\}$ and $Q_4 = \{\mathrm{T}, \mathrm{F}\}$. $\Sigma = \{a, b, c\}$. A represents the set $S \subseteq \Sigma^4$ of strings where

$$S = \{aaaa, aaba, aaca, abaa, abba, abca, acaa, acba, acca, baab, baba, baca, bbab,$$
$$bbba, bbca, bcaa, bcba, bcca, caab, caba, caca, cbaa, cbba, cbca, ccab, ccba, ccca\}$$

The use of a MA for the state store in a reachability analysis requires the partitioning of each state vector (Fig. 1). Here a state vector is partitioned as a sequence of bytes and each byte is allocated to a distinct layer in the automaton. With this approach the ordering of components within the state vector can have

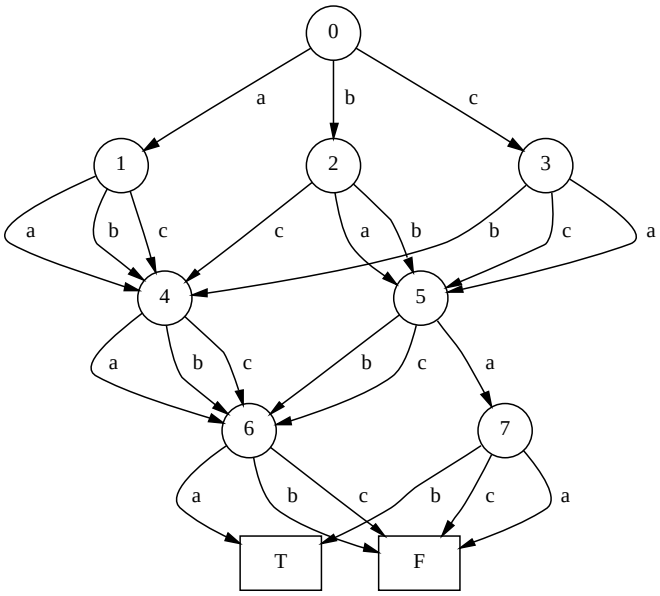

Fig. 3. A minimized automaton

a significant impact upon the space reductions achieved. The key observation concerning the suitability of MA's in the analysis of timed systems is that the clock constraint component Z of a symbolic state (q, Z) is usually represented by a difference bound matrix (DBM) [10] which is also encoded naturally as a byte sequence and so can be easily incorporated into the MA state store. This contrasts significantly with approaches based upon a BDD representation of the state store where this integration is much more problematic.

5 Implementation and Experimental Results

5.1 Implementation

We have implemented a prototype 'compiler' for *bCANDLE*. The compiler is written in ML and generates C code to perform a given task on a system model. The user can choose to generate the timed graph for the model (in KRONOS `.tg` format), or to perform reachability analysis of the simulation graph on-the-fly, without first generating the timed graph, or to explore the simulation graph interactively. In order to experiment with a variety of state space storage modes, we generate C code for the following modes (the mode is chosen as a compile-time option) :

H Each state vector is encoded as a fixed length byte array. In each state, storage is allocated for the maximum number of clocks required system-wide, even

though there may be many states in which fewer clocks are active [9]. The set of visited states is stored in a single hash table.

M As for **H**, except that the set of visited states is stored as a MA.

HV Each state vector is encoded as a fixed length byte array which includes a pointer to a variable dimension matrix [21] giving the zone associated with the state. State vectors are stored in one hash table, variable dimension matrices in another. This means that for each state, only sufficient storage is allocated for the number of clocks active in it, and that only one copy of each distinct zone is stored for the whole system. This corresponds closely to the storage mode adopted by the most recent implementations of KRONOS [21].

MV As for **HV**, except that the hash table storing the state vectors is replaced by a MA.

5.2 Experiments

We have tested our implementation on some example system models: *Boiler1* and *Boiler2* model part of a boiler control system; *Disbmut* models a CAN implementation of a standard algorithm for distributed mutual exclusion [20], the model included here is for a single coordinator and three competing processes. Table 1 gives information regarding the scale of the examples: number of processes, variables, CAN channels, message types and clocks required by each system, and the number of distinct zones and symbolic states identified in generating the whole of the reachable state space in the simulation graph.

System	Procs	Vars	Chans	Mtypes	Clocks	Zones	States
Boiler1	2	2	1	1	5	29073	143802
Boiler2	2	3	1	1	3	581	825643
Disbmut	4	10	1	6	6	71010	194491

Table 1. Test systems

Performance measurements for each system and each state space storage mode are given in Tab. 2. We show the time taken and the total memory used in generating the reachable state space. We take mode **H** as the basis of comparison and show memory compression and time overheads as percentages of the requirements of mode **H**. The measurements were taken on a 233MHz Pentium II having 64Mb RAM (58Mb available) and 128Mb swap, running RedHat Linux 5.0.

In common with other symbolic approaches such as BDD's [7] and GE-sets [12], MA's are sensitive to variable ordering. We have begun to investigate this phenomenon, in our implementation, by permuting the order of the major components of a state vector: marking M, context C (network and data environment) and zone Z. We have generated the reachable state space for each permutation and each of the modes **M** and **MV**.

System	Mode	Mem (Mb)	Comp %	Time (s)	Over %
Boiler1	H	14.57	100	21	100
	M	8.40	58	213	1014
	HV	6.91	47	20	95
	MV	3.12	21	32	152
Boiler2	H	40.88	100	59	100
	M	3.69	9	145	246
	HV	18.79	46	60	102
	MV	4.22	10	105	178
Disbmut	H	32.79	100	74	100
	M	12.24	37	315	426
	HV	18.07	55	74	100
	MV	12.52	38	94	127

Table 2. Comparison of storage modes

In addition, for mode **M**, we have investigated the influence of the placement of cells within the encoding of the matrix representing the zone: order **O0** is the standard row-major encoding of a matrix; **O1** removes clock names from the diagonal, stores them before all other cells and then follows row-major ordering of the remaining cells; finally, **O2** also removes clock names from the diagonal, as for **O1**, and additionally, stores contiguously both the lower and upper bounds for each clock, giving a representation somewhat similar to a CDD [5] in which all constraints are stored.

Tab. 3 shows the state space usage of the orderings which show the best, and the worst, performance for each system and each MA mode. The nodes (resp. edges) column shows the total number of nodes (resp. edges) used in the final MA.

System	Mode	Order	Nodes	Edges	Mem (Mb)
Boiler1	M	CMZ, O1	158861	479927	8.40
	M	CMZ, O2	266451	757426	12.90
	MV	MZC	5250	59940	3.12
	MV	CZM	5693	91865	4.33
Boiler2	M	CZM, O0	50987	206788	3.69
	M	ZCM, O0	602525	1577930	26.62
	MV	CMZ	44300	191575	4.22
	MV	ZCM	594544	1560000	27.37
Disbmut	M	CMZ, O1	226001	594846	12.24
	M	MZC, O0	1454128	3660000	63.74
	MV	CMZ	99838	433329	12.52
	MV	ZCM	223085	674899	18.15

Table 3. Impact of variable ordering on minimized automaton modes

5.3 Discussion of experimental results

Reference to Tab. 2 shows that the most economical use of space, for all examples, involves the use of a MA. However, including the zone encoding in the MA (mode **M**) gives only marginally better use of space than that given by the use of variable dimension matrices (mode **MV**), and this only in two cases; in the other case, mode **MV** is better. This suggests that the use of a MA does not offer a particularly efficient representation of a union of zones but *is* effective in encoding the discrete variables. It remains to be seen if this observation is repeated as we tackle larger systems.

As expected, we pay a time performance penalty for the use of MA; however the better the compression achieved, the less the time penalty. Achieving good compression requires the use of a good variable ordering. From Tab. 3, we observe that in 4 of 6 cases the best ordering for the state vector components is *context, marking, zone* (**CMZ**), and in 2 of 3 cases the best ordering for cell elements is **O1**. Further experiments are needed to see if this is sustained. We also notice that use of a bad variable ordering can be disastrous, as witnessed by the worst case for *Disbmut*, mode **M**.

6 Related work

Representation of timing constraints by DBMs was proposed by Dill [10] and has been preferred in the most efficient verification tools for timed systems, such as KRONOS [13] and UPPAAL [18].

Wong-Toi and Dill [23] and Balarin [4] have each shown techniques for encoding the transition relation of timed systems using BDD's, approximating unions of zones using convex hulls. Bozga et. al. [6] offer a canonical representation of discretized sets of clock configurations using NDDs[2] [3], which are a BDD-based encoding amenable to combination with a symbolic representation of the discrete part of the system.

Larsen et. al. [17] propose a compact encoding for DBMs which provides a minimal and canonical representation of clock constraints and allows for efficient inclusion checking between constraint systems. They do not consider how this representation may be combined with a symbolic representation of the rest of the system.

Behrmann et. al. [5] have recently proposed clock difference diagrams as a data structure for the compact representation of unions of zones. On a variety of case studies, they report space savings of between 46%–99% over their earlier DBM implementation.

7 Conclusions and further work

In this paper, we have proposed the use of MA's for the representation of the state space in the reachability analysis of timed systems. The advantage of this

[2] Numerical Decision Diagrams

approach is that a symbolic representation of the discrete state variables can be combined very simply with a DBM representation of zones. Preliminary experimental results suggest that this leads to significant space reductions which are achieved mainly because of the compact encoding of the discrete variables. The impact of MA's on the space requirements of unions of zones seems less promising. For this reason, we expect to see the greatest benefits in the analysis of asynchronous, data-bearing systems where the value of this approach has already been demonstrated in untimed settings [11, 12]; fortunately for us, CAN-based systems mainly fall within this class.

For the future, it would be interesting to investigate the combination of MA's with CDD's. We expect also that an MA option would be a useful addition to KRONOS and UPPAAL now that both tools handle system descriptions with discrete variables.

Acknowledgements We are grateful to Stavros Tripakis, Sergio Yovine, Jean-Charles Grégoire and Gerard Holzmann for generously making their code available to us.

References

1. L. Aceto, A. Burgueño, and K. Larsen. Model checking via reachability testing for timed automata. Technical report, BRICS,Aarhus University, 1997.
2. R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–236, 1994. Preliminary version appears in Proc. 17th ICALP, 1990, LNCS 443.
3. 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, *Proc. 1st Int. Workshop on Hybrid and Real-Time Systems (HART'97)*, volume 1201 of *Lecture Notes in Computer Science*, pages 346–360. Springer Verlag, March 1997.
4. F. Balarin. Approximate reachability analysis of timed automata. In *Proc. of 17th IEEE Real Time Systems Symposium*, pages 52–61. IEEE Computer Society Press, 1996.
5. G. Behrmann, K. Larsen, J. Pearson, C. Weise, and W. Yi. Efficient timed reachability analysis using clock difference diagrams. In *Proceedings of the 11th International Conference on Computer Aided Verification (CAV'99)*, Lecture Notes in Computer Science. Springer Verlag, 1999. (to appear).
6. M. Bozga, O. Maler, A. Pnueli, and S. Yovine. Some progress in the symbolic verification of timed automata. In O. Grumberg, editor, *Proceedings of the 9th International Conference on Computer Aided Verification (CAV'97)*, volume 1254 of *Lecture Notes in Computer Science*, pages 179–190, Haifa, Israel, June 1997. Springer Verlag.
7. R. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 8(C-35):677–691, 1986.
8. C. Daws and S Tripakis. Model checking of real-time reachability properties using abstractions. In B. Steffen, editor, *Proceedings of 3nd International Workshop on Tools and Algorithms for the Construction of Systems (TACAS'98)*, volume 1384 of *Lecture Notes in Computer Science*. Springer Verlag, 1998.

9. C. Daws and S. Yovine. Reducing the number of clock variables of timed automata. In *Proc. of 17th IEEE Real-Time Systems Symposium*, December 1996.
10. D. 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*, pages 197–212. Springer Verlag, 1989.
11. F. Gagnon, J.-C. Grégoire, and D. Zampunieris. Sharing tree for "on-the-fly" verification. In *Proc. Int. Conf. on Formal Description Techniques VIII (FORTE'95)*. IEEE Computer Society Press, 1995.
12. J.-Ch. Grégoire. State space compression in SPIN with GE-sets. In *Proc. 2nd SPIN Workshop, Rutgers University, New Jersey, USA*, August 1996.
13. T. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111(2):193–244, 1994.
14. G. Holzmann and A. Puri. A minimized automaton representation of reachable states. In *Software Tools for Technology Transfer*. Springer Verlag, 1999. (to appear).
15. ISO/DIS 11898: Road Vehicles – interchange of digital information – Controller Area Network (CAN) for high speed communication, 1992.
16. D. Kendall, S. Bradley, W.D. Henderson, and A.P. Robson. *bCANDLE*: Formal modelling and analysis of CAN control systems. In *Proceedings of 4th IEEE Real Time Technology and Applications Symposium (RTAS'98)*, pages 171–177. IEEE Computer Society Press, June 1998.
17. K. Larsen, F. Larsson, P. Pettersson, and Wang Yi. Efficient verification of real-time systems: Compact data structure and state-space reduction. In *Proc. of 18th IEEE Real Time Systems Symposium*, December 1997.
18. K. Larsen, P. Pettersson, and Wang Yi. UPPAAL in a Nutshell. *Springer International Journal on Software Tools for Technology Transfer*, October 1997.
19. X. Nicollin and J. Sifakis. The algebra of timed processes, ATP: Theory and application. *Information and Computation*, 114:131–178, 1994.
20. A. Tanenbaum. *Modern Operating Systems*. Prentice Hall International, 1992.
21. S. Tripakis. *The Formal Analysis of Timed Systems in Practice*. PhD thesis, Université Joseph Fourier, Grenoble, December 1998.
22. W. Visser. Memory efficient state storage in SPIN. In *Proc. 2nd SPIN Workshop, Rutgers University, New Jersey, USA*, August 1996.
23. H. Wong-Toi and D. Dill. Approximations for verifying timing properties. In T. Rus and C. Rattray, editors, *Theories and Experiences for Real-time System Development*. World Scientific Publishing, 1994.
24. S. Yovine. *Méthodes et Outils pour la Vérification Symbolique de Systèmes Temporisés*. PhD thesis, Institut National Polytechnique de Grenoble, May 1993.