Catholic University in Ružomberok
Scientific Issues, Mathematica II, Ružomberok 2008

A SAT-based technique to analyse networks of finite automata

Bożena Woźna, Andrzej Zbrzezny

Institute of Mathematics and Computer Science
Jan Długosz University of Częstochowa
al. Armii Krajowej 13/15, 42-200 Częstochowa, Poland
e-mail: {b.wozna, a.zbrzezny}@ajd.czest.pl

Abstract
We present a SAT-based technique to analyse properties of a network of finite automata. In particular, we show how to encode any finite transition system via Boolean formulae, and we present a reachability algorithm, which can be easily extended to work with more complicated sort of automata (for example, timed automata). We exemplify the use of the technique by means of a classic problem - "mutual exclusion".

1. Introduction

Model checking [3] is an automatic verification technique, which does not rely on complicated interaction with the user for incremental property proving. If a property does not hold, the model checker automatically generates a counterexample. In model checking, the system to be verified is modelled as a finite state machine (for example a network of finite automata), which further can be represented by a transition system, and temporal logics are used for specifying the system properties. Typical examples of finite systems, for which model checking has successfully been applied, are digital sequential circuits and communication protocols, and typical examples of checked properties are reachability properties.

Unfortunately, the practical applicability of model checking is strongly restricted by the state explosion problem, which is mainly caused by representing concurrency of operations by their interleaving. Therefore, there are many different reduction techniques aimed at minimising models. The major methods include application of partial order reductions [9, 11], symmetry reductions [6], abstraction techniques [5], BDD-based symbolic methods [2], and SAT-based techniques that include both the bounded (BMC) [1, 10] and unbounded (UMC) [7] model checking methods.

In this paper we make an attempt to show how to use the bounded model checking technique to check reachability properties of a finite state systems modelled via a network of finite automata.

The rest of the article is structured as follows. In the next section we give a technical introduction to finite automata and their semantics. In this section we also describe the well-known mutual exclusion problem and depict its automata model. In the following section we show Boolean encoding of finite transition systems. We exemplify this encoding by means of the mutual exclusion problem. In section 4 we describe a reachability analysis that is based on the bounded model checking technique. In the following section we discuss experiments performed for the mutual exclusion problem modelled by network of finite automata. Finally, we detail our conclusions.

2. Finite automata

An automaton is a tuple $A = (\Sigma, L, l^0, \delta)$ where:

- Σ is a finite alphabet;
- L is a finite set of locations, and $l^0 \in L$ is an initial location;
- $\delta \subseteq L \times \Sigma \times L$ is a transition relation;

An automaton can be represented as a graph; for instance, Figure 1 depicts three automata. The first one (called *the process 1*) is characterised by $\Sigma = \{a_1, in_1, out_1\}$, $L = \{l_0, l_1, l_2\}$, $l^0 = l_0$ (this is indicated with an incoming arrow in the graph), and $\delta = \{(l_0, a_1, l_1), (l_1, in_1, l_2), (l_2, out_1, l_0)\}$. The second one (called *the process 2*) is characterised by $\Sigma = \{a_2, in_2, out_2\}$, $L = \{l_0, l_1, l_2\}$, $l^0 = l_0$ (this is indicated with an incoming arrow in the graph), and $\delta = \{(l_0, a_2, l_1), (l_1, in_2, l_2), (l_2, out_2, l_0)\}$. The third one (called *the permission process*) is characterised by $\Sigma = \{in_1, in_2, out_1, out_2\}$, $L = \{l_0, l_1\}$, $l^0 = \{l_0\}$ (this is indicated with an incoming arrow in the graph), and $\delta = \{(l_0, in_1, l_1), (l_0, in_2, l_1), (l_1, out_1, l_0), (l_1, out_2, l_0)\}$.

2.1 Network of automata

In general, a system consists of several automata forming a network that run in parallel and communicate with each other via executing joint actions. There are several ways of defining this communication (known as *composition of automata*), but our definition determines the *multi-way synchronisation*, i.e., it requires that each component containing a transition labelled by a joint action has to execute it in order the system can move along that action. For instance, Figure 1 depicts a net-

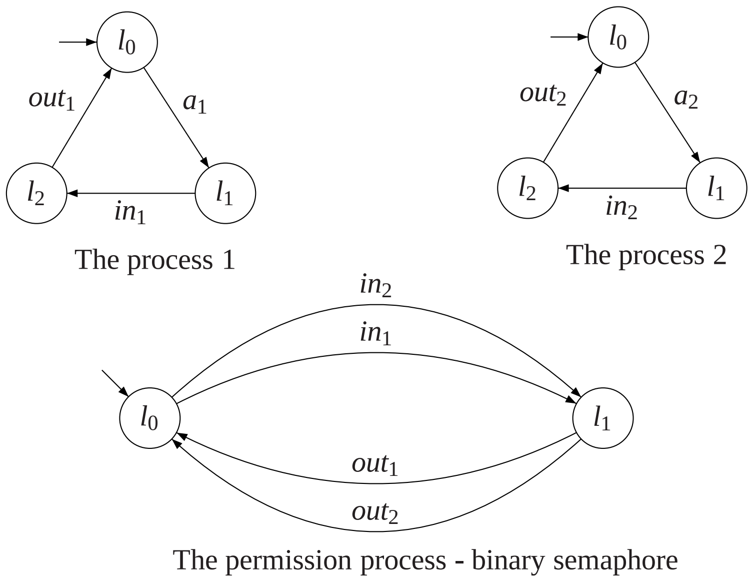

Figure 1: A network of automata - Mutual Exclusion.

work that consists of three automata which communicate via the following actions: $\{in_1, in_2, out_1, out_2\}$.

The following definition formalises the above discussion. Let $A_i = (\Sigma_i, L_i, l_i^0, \delta_i)$ be an automaton, for $i = 1, \ldots, m$, $\Sigma = \bigcup_{i=1}^{m} \Sigma_i$, and let for $\sigma \in \Sigma$, the set $\Sigma(\sigma) = \{1 \leq i \leq m \mid \sigma \in \Sigma_i\}$ gives the numbers of the components that synchronise at σ.

Definicja 1. A composition of m automata A_i, *for* $i = 1, \ldots, m$, *is the automaton* $A = (\Sigma, L, l^0, \delta)$, *where*

- $\Sigma = \bigcup_{i=1}^{m} \Sigma_i$, $L = \prod_{i=1}^{m} L_i$, $l^0 = (l_1^0, \ldots, l_m^0)$, *and*
- *a transition* $((l_1, \ldots, l_m), \sigma, (l'_1, \ldots, l'_m)) \in \delta$ *iff* $(\forall j \in \Sigma(\sigma))\ (l_j, \sigma, l'_j) \in \delta_j$ *and* $(\forall i \in \{1, \ldots, m\} \setminus \Sigma(\sigma))\ l'_i = l_i$.

Figure 2 illustrates the above definition.

2.2 Semantics

The semantics of an automaton A is defined by associating to it a *transition system* as defined below.

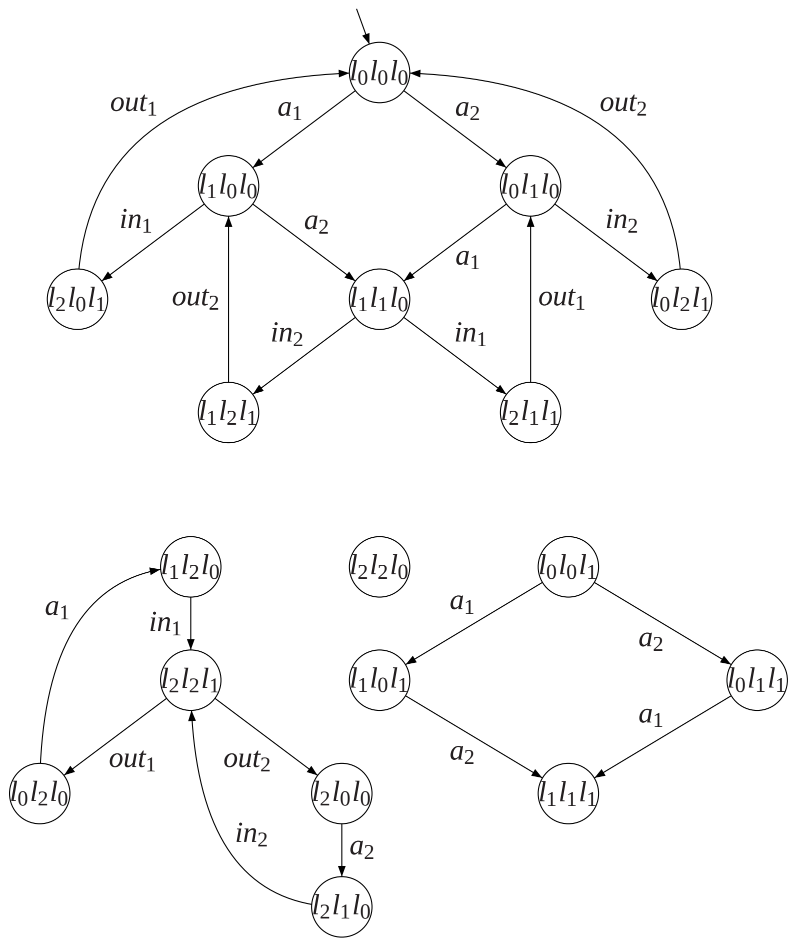

Figure 2: A composition of the three automata depicted in Figure 1.

Definicja 2 (transition system)**.** *Let* $A = (\Sigma, L, l^0, \delta)$ *be an automaton. A* transition system *for* A *is a tuple* $TS = (S, \iota, T)$ *where* $S = L$ *is a set of states,* $\iota = l^0$ *is an initial state, and* $T \subseteq S \times S$ *is a transition relation defined by action transitions:* $(s, s') \in T$ *iff there exists* $\sigma \in \Sigma$ *such that* $(s, \sigma, s') \in \delta$*.*

2.3 Mutual exclusion problem

Now we describe **a mutual exclusion problem**, the well-known example in the literature of distributed systems; its automata model is shown in Figure 1.

The system consists of $n + 1$ processes for $n \geq 2$: the processes $1, \ldots, n$, and the permission process. The processes $1, \ldots, n$ compete for an access to the same shared resource. At any moment of time at most one of these processes can use the resource.

It is the task of the permission process to schedule an access to the resource.

The availability of the resource is represented by a "permission" location l_0 in the semaphore. A "critical section" is a part of a process that uses the shared resource and hence needs protection against a possible "disturbance" by the other process. The critical section of the process i (for $i = 1, \ldots, n$) is represented by the location l_2(crit). The non-critical part of the process i is represented by the location l_0 (remainder) and l_1 (wait). The process i can perform the actions in_i, out_i and a_i (entering the critical section, exiting the critical section, and an action outside the critical section).

To enter or exit the critical section it has to synchronise with the permission process, which contains the following locations: l_0 (the resource is not used), l_1 (the resource is used by the process i), for $i = 1, \ldots, n$. The permission process can perform the actions in_i and out_i, $i = 1, \ldots, n$.

3. Boolean Encoding of Transition Systems

To represent a transition system $TS = (S, \iota, T)$ via some Boolean formula, we have to encode its states, in particular its initial state, and the transition relation. For example, assume that we want to encode the transition system for the automata model of the mutual exclusion problem that it is shown in Figure 1. We proceed as follows.

Let $\mathcal{SV} = \{p_1, p_2, \ldots\}$ be an infinite set of propositional variables that are called `state variables`; this is because they are used to encode states of transition systems.

We start with a Boolean encoding of all the local states of all the automata that are shown in Figure 1.

The process 1				*The process* 2			
loc.	bit_2	bit_1	*formula*	*loc.*	bit_4	bit_3	*formula*
l_0	0	0	$\neg p_1 \wedge \neg p_2$	l_0	0	0	$\neg p_3 \wedge \neg p_4$
l_1	0	1	$\neg p_1 \wedge p_2$	l_1	0	1	$\neg p_3 \wedge p_4$
l_2	1	0	$p_1 \wedge \neg p_2$	l_2	1	0	$p_3 \wedge \neg p_4$

The permission process		
location	bit_5	*formula*
l_0	0	$\neg p_5$
l_1	1	p_5

Given the above, it is easy to see that each state of a transition system TS that is associated to the network of automata, which models the mutual exclusion problem, can be represented by a valuation of a vector $\mathbf{w} = (p_1, \ldots, p_5)$, called a *global state variable*.

Then, a Boolean formula $I(\mathbf{w})$, which encodes the initial global state of the considered system is the conjunction of three formulae that encode all the local initial states, i.e.,:

$$I(\mathbf{w}) = (\neg p_1 \wedge \neg p_2) \wedge (\neg p_3 \wedge \neg p_4) \wedge (\neg p_5)$$

Furthermore, let $\mathbf{w} = (p_1, \ldots, p_5), \mathbf{w}' = (p'_1, \ldots, p'_5)$ be two different global states variables. A Boolean formula $\mathcal{T}(\mathbf{w}, \mathbf{w}')$, which encodes all the transitions of the considered system is defined as the disjunction of formula that encode single transitions:

a_1	$\neg p_1 \wedge \neg p_2 \wedge \neg p'_1 \wedge p'_2 \wedge (p_3 \leftrightarrow p'_3) \wedge (p_4 \leftrightarrow p'_4) \wedge (p_5 \leftrightarrow p'_5)$
	$\vee$
in_1	$\neg p_1 \wedge p_2 \wedge p'_1 \wedge \neg p'_2 \wedge (p_3 \leftrightarrow p'_3) \wedge (p_4 \leftrightarrow p'_4) \wedge \neg p_5 \wedge p'_5$
	$\vee$
out_1	$p_1 \wedge \neg p_2 \wedge \neg p'_1 \wedge \neg p'_2 \wedge (p_3 \leftrightarrow p'_3) \wedge (p_4 \leftrightarrow p'_4) \wedge p_5 \wedge \neg p'_5$
	$\vee$
a_2	$\neg p_3 \wedge \neg p_4 \wedge \neg p'_3 \wedge p'_4 \wedge (p_1 \leftrightarrow p'_1) \wedge (p_2 \leftrightarrow p'_2) \wedge (p_5 \leftrightarrow p'_5)$
	$\vee$
in_2	$\neg p_3 \wedge p_4 \wedge p'_3 \wedge \neg p'_4 \wedge (p_1 \leftrightarrow p'_1) \wedge (p_2 \leftrightarrow p'_2) \wedge \neg p_5 \wedge p'_5$
	$\vee$
out_2	$p_3 \wedge \neg p_4 \wedge \neg p'_3 \wedge \neg p'_4 \wedge (p_1 \leftrightarrow p'_1) \wedge (p_2 \leftrightarrow p'_2) \wedge p_5 \wedge \neg p'_5$

So in general we can say that since the set of states of a transition system TS associated to a network of automata is finite, every element of $S = S_1 \times S_m$ can be encoded as a bit vector of length $r = \sum_{i=1}^{m} \lceil \log_2(|S_i|) \rceil$ depending on the number of states of TS. Therefore each state can be represented by a global state variable $\mathbf{w} = (p_1, \ldots, p_r)$.

The formula $I(\mathbf{w})$ that encodes the initial state of the system is defined as a formula which for every valuation V of states variables satisfies:

$$I(\mathbf{w}) \text{ is true iff } V(\mathbf{w}) = \iota$$

Further, to encode all the transitions of TS we have to take two global state variables: $\mathbf{w} = (p_1, \ldots, p_r)$ and $\mathbf{w}' = (p_{r+1}, \ldots, p_{2r})$, and then define the Boolean formula $\mathcal{T}(\mathbf{w}, \mathbf{w}')$ as a formula that for every valuation V of states variables satisfies:

$$\mathcal{T}(\mathbf{w}, \mathbf{w}') \text{ is true iff } (V(\mathbf{w}), V(\mathbf{w}')) \in T$$

4. Reachability Analysis

Given an automaton A and a set of states P. The *reachability problem* consists in establishing whether a state in P is reachable from the initial state of the transition system associated to A.

4.1 Checking reachability

In order to solve the reachability problem for an automaton $A = (\Sigma, L, l^0, \delta)$, we have to unfold the transition relation of the transition system $TS = (S, \iota, T)$ associated to this automaton to the depth $k \in \mathbb{N}$. For that purpose we first define a *k-path* as a finite sequence $\pi = (s_0, \cdots, s_k)$ of states such that s_0 is the initial state of TS and $(s_i, s_{i+1}) \in T$ for each $0 \le i < k$.

Next, we define a formula $path_k(\mathbf{w}_0, \ldots, \mathbf{w}_k)$ that encodes all the k-paths starting at the initial state of TS:

$$path_k(\mathbf{w}_0, \ldots, \mathbf{w}_k) := I(\mathbf{w}_0) \wedge \bigwedge_{i=0}^{k-1} \mathcal{T}(\mathbf{w}_i, \mathbf{w}_{i+1}) \tag{1}$$

where $\mathbf{w}_i$ are global state variables.

Further, let $\mathcal{P}(\mathbf{w})$ be a Boolean formula that encodes the set of states satisfying some undesired property. Given Formula 1 and the propositional formula $\mathcal{P}(\mathbf{w})$, we try to establish whether it is possible to reach a state that satisfies $\mathcal{P}(\mathbf{w})$ by checking the satisfiability of the following formula:

$$path_k(\mathbf{w}_0, \ldots, \mathbf{w}_k) \wedge \bigvee_{i=0}^{k} \mathcal{P}(\mathbf{w}_i) \tag{2}$$

We stop the unfolding of the transition relation if either Formula 2 is satisfiable (which means that a state we are searching for is reachable), or it is impossible to check the satisfiability of the formula in question for a given SAT-solver. We could also stop if the value of k is equal to the number of states of the transition system in question (which means that a state we are searching for is unreachable).

4.2 Checking unreachability

If the searched state is unreachable in the considered system, then the above method is infeasible for even fairly small systems. Therefore now we describe another technique [12], but this time it is designed for checking the unreachability. For that purpose we first define a *free k-path* to be a finite sequence $\pi = (s_0, \cdots, s_k)$ of states such that $(s_i, q_{i+1}) \in T$ for each $0 \le i < k$.

Next, we define the formula $freepath_k(\mathbf{w}_0, \ldots, \mathbf{w}_k)$ which encodes all the free k-paths:

$$freepath_k(\mathbf{w}_0, \ldots, \mathbf{w}_k) := \bigwedge_{i=0}^{k-1} \mathcal{T}(\mathbf{w}_i, \mathbf{w}_{i+1}) \tag{3}$$

where $\mathbf{w}_i$ are global state variables.

Further, let $\mathcal{P}(\mathbf{w})$ be a Boolean formula that encodes the set of states satisfying some undesired property. We try to establish the length of the longest free $k-$path π

such that the undesired property $\mathcal{P}$ is true in the last state and false in all the previous states of π by checking the satisfiability of the following formula:

$$freepath_k(\mathbf{w}_0, \ldots, \mathbf{w}_k) \wedge \mathcal{P}(\mathbf{w}_k) \wedge \bigwedge_{i=0}^{k-1} \neg\mathcal{P}(\mathbf{w}_i) \tag{4}$$

We stop the unfolding of the transition relation if either Formula 4 is unsatisfiable (which means that we have found the desired length), or it is impossible to check the unsatisfiability of the formula in question for the used SAT-solver. If we have found the desired length, say k_0, it is enough to check the satisfiability of Formula 2 on the paths of the lengths $k < k_0$ only. If Formula 2 is unsatisfiable for every $k < k_0$ we can conclude that the investigated property is unreachable. In fact, it is enough to check the satisfiability of Formula 2 on the path of the length $k = k_0 - 1$ only.

Note that it may be the case that for a given transition system there exist arbitrary long free paths such that the undesired property is true in the last state and false in all the previous states. Therefore our method is incomplete. However, as we shall see, for some systems it works quite efficiently.

5. Experimental results

Our experiments were performed on a computer equipped with the processor Intel Pentium D (3 GHz), 2 GB main memory and the operating system Linux 2.6.24. As the benchmark we used the mutual exclusion problem, described in section 2.3, and we have tested the following mutual exclusion property:

$$\varphi = \bigwedge_{i\in\{1\ldots n\},\ j\in\{1\ldots n\},\ i<j} (\neg crit_i \vee \neg crit_j)$$

which means that certain sections of code (*critical sections*) will not be executed by more than one process simultaneously.

In our experiments we have compared two methods described in section 4. To test the first method (section 4.1) we used the $k-$paths of length equal to the size of the considered model, and it turned out that we were able to check mutual exclusion property for 4 processes only. To test the second method (section 4.2), we applied the algorithm described in [12]. This time it turned out that we were able to check mutual exclusion property for 12 processes. The results for the method are summarised in Table 1. Note that in this case the checked model has 1 062 882 states.

To verify satisfaction of φ for 4 processes, using the first method, we checked satisfaction of the Boolean formula encoding the translation of the formula φ on $k-$paths of length 162. The formula in question consists of 17538 variables and 48205 clauses; our implementation needed 0.12 second and 4.6 MB memory to produce it. Its unsatisfiability was checked by RSat version 2.01 [8], a mainstream SAT solver; 856.6 seconds and 122.4 MB of memory were needed to check this.

		BMC				RSat		
k	**path**	**variables**	**clauses**	**sec**	**MB**	**sec**	**MB**	**satisfiable**
0	initial	218	555	0.0	1.4	1.3	0.0	NO
1		693	1905	0.0	1.5	1.4	0.0	YES
1	initial	706	1944	0.0	1.5	1.4	0.0	NO
2		1171	3262	0.0	1.5	1.5	0.0	YES
2	initial	1182	3297	0.0	1.5	1.5	0.0	NO
3		1647	4615	0.0	1.7	1.5	0.0	YES
3	initial	1658	4650	0.0	1.7	1.7	0.1	NO
4		2125	5972	0.0	1.7	1.7	0.0	YES
4	initial	2134	6003	0.0	1.7	1.8	0.1	NO
5		2601	7325	0.0	1.8	1.8	0.0	YES
5	initial	2610	7356	0.0	1.8	2.1	0.3	NO
6		3079	8682	0.0	1.9	1.9	0.0	YES
6	initial	3086	8709	0.0	1.9	2.7	0.6	NO
7		3555	10035	0.0	1.9	2.0	0.0	YES
7	initial	3562	10062	0.0	1.9	3.3	1.5	NO
8		4033	11392	0.0	2.0	2.1	0.0	YES
8	initial	4038	11415	0.0	2.0	3.6	1.9	NO
9		4509	12745	0.0	2.0	2.2	0.0	YES
9	initial	4514	12768	0.0	2.0	3.8	1.8	NO
10		4987	14102	0.0	2.2	2.5	0.1	YES
10	initial	4990	14121	0.0	2.2	4.2	3.2	NO
11		5463	15455	0.0	2.2	2.5	0.0	YES
11	initial	5466	15474	0.0	2.2	7.0	7.4	NO
12		5941	16812	0.0	2.3	4.5	1.3	YES
12	initial	5942	16827	0.0	2.3	8.6	6.7	NO
13		6417	18165	0.0	2.4	177.8	4442.9	NO
			In total:	0.6	2.4	246.4	4442.9	

Table 1: The preservation of φ. The second method.

6. Final remarks

The main point that we wished to bring across here is that the reachability analysis based on the direct application of the BMC method is infeasible even for fairly small systems. However, if we apply a backward analysis that use the BMC method, then the reachability checking can be done for fairly large systems.

Further, the experiments confirm that the second method leads to a reduction on the size of the CNF formulae submitted to the SAT solver, and to a significant reduction both in the time and in the memory required by the SAT solver to return an answer.

References

[1] A. Biere, A. Cimatti, E. Clarke, M. Fujita, and Y. Zhu. Symbolic model checking using SAT procedures instead of BDDs. In *Proceedings of the ACM/IEEE Design Automation Conference (DAC'99)*, pages 317–320, 1999.

[2] R. Bryant. Binary Decision Diagrams and beyond: Enabling technologies for formal verification. In *Proceedings of ICCAD'95*, pages 236–243, 1995.

[3] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. The MIT Press, Cambridge, Massachusetts, 1999.

[4] F. Copty, L. Fix, R. Fraer, E. Giunchiglia, G. Kamhi, A. Tacchella, and M. Vardi. Benefits of bounded model checking at an industrial setting. In *Proceedings of CAV'01*, volume 2102 of *LNCS*, pages 436–453. Springer-Verlag, 2001.

[5] D. Dams, O. Grumberg, and R. Gerth. Abstract interpretation of reactive systems: Abstractions preserving ACTL*, ECTL* and CTL*. In *Proceedings of PROCOMET'94*. Elsevier Science Publishers, 1994.

[6] E. A. Emerson and C. S. Jutla. Symmetry and model checking. In *Proceedings of CAV'93*, volume 697 of *LNCS*, pages 463–478. Springer-Verlag, 1993.

[7] K. L. McMillan. Applying SAT methods in unbounded symbolic model checking. In *Proceedings of CAV'02*, volume 2404 of *LNCS*, pages 250–264. Springer-Verlag, 2002.

[8] K. Pipatsrisawat and A. Darwiche. RSat 2.0: SAT Solver Description. *Technical report of Automated Reasoning Group*, Computer Science Department, UCLA, Nr D-153, 2007.

[9] W. Penczek, M. Szreter, R. Gerth, and R. Kuiper. Improving partial order reductions for universal branching time properties. *Fundamenta Informaticae*, 43:245–267, 2000.

[10] W. Penczek, B. Woźna, and A. Zbrzezny. Bounded model checking for the universal fragment of CTL. *Fundamenta Informaticae*, 51(1-2):135–156, 2002.

[11] P. Wolper and P. Godefroid. Partial order methods for temporal verification. In *Proceedings of the 4th International Conference on Concurrency Theory (CONCUR'93)*, volume 715 of *LNCS*, pages 233–246. Springer-Verlag, 1993.

[12] A. Zbrzezny. Improvements in SAT-based reachability analysis for timed automata. *Fundamenta Informaticae*, 60(1-4):417–434, 2004.