Minimising Deterministic Büchi Automata Precisely using SAT Solving*

Rüdiger Ehlers

Reactive Systems Group
Saarland University, Germany

Abstract. We show how deterministic Büchi automata can be fully minimised by reduction to the satisfiability (SAT) problem, yielding the first automated method for this task. Size reduction of such ω-automata is an important step in probabilistic model checking as well as synthesis of finite-state systems. Our experiments demonstrate that state-of-the-art SAT solvers are capable of solving the resulting satisfiability problem instances quickly, making the approach presented valuable in practice.

1 Introduction

The success of techniques for formal verification is closely connected to advances in the efficient solution of the satisfiability (SAT) problem. In the area of *bounded model checking* [4], it has been shown that checking whether a given system has a bounded-length witness for erroneous behaviour can efficiently and effectively be performed by reduction to the SAT problem. Likewise, in software verification, approaches involving *satisfiability modulo theory* solvers [2] have emerged, which rely on the underlying SAT techniques.

However, there are many areas in formal verification that do not benefit from advances in SAT solving yet. Taking for example probabilistic model checking, where many of its variants are PSPACE-hard or require computing some quantitative result [1], it is by no means obvious how the success of modern SAT techniques can be transferred to this area. In this paper, we present some progress towards closing this gap by reporting how SAT solving can be used for the full minimisation of deterministic automata over infinite words, which arise in intermediate steps of many formal methods.

One particular application of this automaton type is the verification of *Markov decision processes* against properties stated in *linear time temporal logic* (LTL) [1]. Here, the classical model checking procedure requires converting the LTL formula to a deterministic automaton. A similar situation arises in common approaches to *synthesis of finite state systems* from LTL specifications [15].

* This work was supported by the German Research Foundation (DFG) within the program "Performance Guarantees for Computer Systems" and the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems" (SFB/TR 14 AVACS).

In both application areas, the minimisation of the number of states of the deterministic automata involved results in significant speed-ups of the actual model checking and synthesis tasks. Current tools for computing automata equivalent to LTL formulas use state space reduction techniques such as *bisimulation quotienting* [8] that have been developed for the minimisation of *non-deterministic* Büchi automata which are suitable for model checking non-deterministic systems. As the universality problem, and thus, the minimisation problem, of this class of automata is PSPACE-hard, the techniques used are typically incomplete and thus do not guarantee to find a minimal automaton. To the best of our knowledge, special techniques for the minimisation of deterministic automata have only been considered for *weak deterministic automata* [13], which however cannot even represent basic liveness properties.

In this paper, we lift the applicability of (complete) automaton minimisation to a more expressive class of deterministic automata. For conciseness, we restrict ourselves to deterministic *Büchi* automata (DBA) here. While DBA are not expressive enough to represent all LTL specifications (and are strictly less expressive than for example deterministic parity automata), they can represent most properties that appear in hardware specifications [6] and it is assured that in these cases, the smallest DBA is not larger than the smallest deterministic parity, Rabin or Streett automaton for the given property [12].

In both application areas mentioned above, the specification is usually a Boolean combination of a set of individual properties, which allows the precise minimisation of the specification parts representable as DBA with our technique before the automaton for the overall specification is composed. Additionally, some modern approaches for fast synthesis from LTL properties achieve a remarkable speedup by even requiring the specification parts to be representable and given as DBAs (in a certain encoding) [14].

In the following, we show that the problem of deciding whether there exists a smaller deterministic Büchi automaton for some given such automaton is contained in NP and can efficiently be reduced to the SAT problem. We evaluate whether currently available SAT solvers are capable of dealing with such problem instances. By succesively building and solving the resulting SAT instances, the minimal automaton is found. Our experiments suggest that for practical applications, the current state of the art in SAT solving is sufficient for the successful usage of the this minimisation technique.

2 Preliminaries

A deterministic Büchi automaton (DBA) is a 5-tuple $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ with a finite set of states Q, a finite alphabet Σ, a transition function $\delta : Q \times \Sigma \to Q$, an initial state $q_0 \in Q$ and a set of accepting states $F \subseteq Q$. For the scope of this paper, we assume that δ is a *total* function.

We say that some infinite word $w = w_0 w_1 \ldots \in \Sigma^\omega$ is accepted by $\mathcal{A}$ if the run π induced by w on $\mathcal{A}$ is *accepting*. We define $\pi = \pi_0 \pi_1 \ldots$ such that $\pi_0 = q_0$ and for all $i \in \mathbb{N}_0$, $\delta(\pi_i, w_i) = \pi_{i+1}$. We say that π is accepting if and only if

$\{q \in Q \mid \exists^\infty j \in \mathbb{N} : \pi_j = q\} \cap F \neq \emptyset$, i.e., some accepting state occurs infinitely often on π. We define the language $\mathcal{L}(\mathcal{A})$ of $\mathcal{A}$ to consist of all words in Σ^ω that induce accepting runs. We denote the size of an automaton by $|Q|$. A DBA is minimal if there exists no other DBA with the same alphabet that has less states and accepts the same language. We define the language of a state $q \in Q$ to be $\mathcal{L}(q) = \mathcal{L}(\mathcal{A}_q)$ for $\mathcal{A}_q = (Q, \Sigma, \delta, q, F)$.

For space reasons, we do not describe the logic LTL here, but rather refer to [4, 1]. A word can either satisfy an LTL formula or not. We define the language of an LTL formula to be the set of infinite words over $\Sigma = 2^{\mathsf{AP}}$ satisfying the formula over some set of atomic propositions AP. We call a DBA equivalent to an LTL formula if their languages are the same.

3 Minimising deterministic Büchi automata

Minimising deterministic Büchi automata is different from minimising deterministic automata over finite words: while for the latter, there exists a suitable polynomial algorithm, which is based on merging states with the same *language*, the same idea cannot be exploited for Büchi automata. The left part of Figure 1 shows a Büchi automaton over the alphabet $\Sigma = 2^{\{a,b\}}$ equivalent to the LTL formula $(\mathsf{GF}a) \wedge (\mathsf{GF}b)$. This DBA has four states, whereas the smallest Büchi automata equivalent to this formula have only three states. One such DBA is depicted in the right part of Figure 1. It is by no means obvious how to *restructure* the left automaton in order to obtain such a smaller one.

This example shows that we cannot only rely on language equivalence for minimising deterministic Büchi automata. We thus propose a different approach here. Assume that some n-state *reference* automaton $\mathcal{A}' = (Q', \Sigma, \delta', q_0', F')$ is given. We use a SAT solver to consider all possible $n-1$-state *candidate automata* $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ and encode the equivalence check of the languages of $\mathcal{A}$ and $\mathcal{A}'$ in clausal form. While such a check is PSPACE-complete for non-deterministic Büchi automata, it can be performed in polynomial time for deterministic Büchi

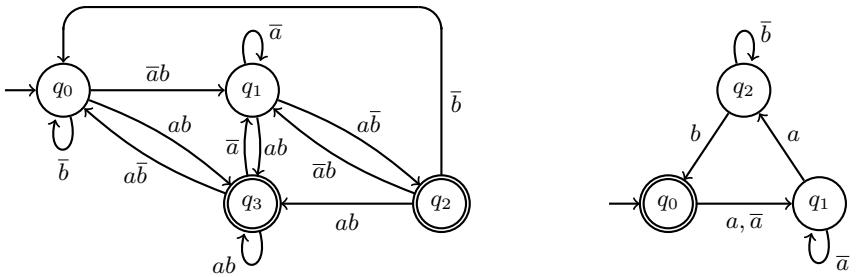

Fig. 1. A non-minimal DBA (left) and a minimal DBA (right) equivalent to the LTL formula $(\mathsf{GF}a) \wedge (\mathsf{GF}b)$ over the atomic proposition set $\mathsf{AP} = \{a, b\}$. Both DBA have a total transition relation. Accepting states are doubly-circled.

automata and can also be efficiently encoded into a SAT instance. By repeatedly applying this reduction technique until the resulting SAT instance becomes unsatisfiable, we obtain an automaton of minimal size.

For encoding the equivalence check, we use the observation that we can deduce from the *product* of $\mathcal{A}$ and $\mathcal{A}'$ if the two deterministic Büchi automata have the same language. More precisely, we build the graph $G = \langle V, E\rangle$ with the set of vertices $V = Q \times Q'$ and edges $E = \{((q_1, q_1'), (q_2, q_2')) \mid \exists s \in \Sigma : \delta(q_1, s) = q_2 \wedge \delta'(q_1', s) = q_2'\}$. If there is some loop $(v_0, v_0')(v_1, v_1') \dots (v_k, v_k')$ in $\langle V, E\rangle$ with (v_0, v_0') being reachable from (q_0, q_0') such that on it, accepting states are only visited for $\mathcal{A}$, but not for $\mathcal{A}'$, i.e., $\{v_0, \dots, v_k\} \cap F \neq \emptyset$ and $\{v_0', \dots, v_k'\} \cap F' = \emptyset$, then there exists some $w \in \mathcal{L}(\mathcal{A})$ such that $w \notin \mathcal{L}(\mathcal{A}')$, so $\mathcal{A}$ and $\mathcal{A}'$ are inequivalent. Dually, if $\{v_0, \dots, v_k\} \cap F = \emptyset$ and $\{v_0', \dots, v_k'\} \cap F' \neq \emptyset$, then there exists a word $w \notin \mathcal{L}(\mathcal{A})$ such that $w \in \mathcal{L}(\mathcal{A}')$. If no such loops can be found, $\mathcal{A}$ and $\mathcal{A}'$ are equivalent.

3.1 Encoding as a SAT problem

Using the observation stated above, we can build a SAT problem instance for solving the DBA state reduction problem. For notational convenience, in the following, primed state variables always refer to states in the reference automaton, whereas unprimed state variables refer to the candidate automaton. We use the following set of variables:

$$\{\langle q\rangle_F \mid q \in Q\} \cup \{\langle q_1, s, q_2\rangle_\delta \mid q_1, q_2 \in Q, s \in \Sigma\} \cup \{\langle q, q'\rangle_G \mid q \in Q, q' \in Q'\}$$
$$\cup \{\langle q_1, q_1', q_2, q_2'\rangle_X \mid q_1, q_2 \in Q, q_1', q_2' \in Q', X \in \{N, A\}\}$$

The SAT clauses are defined as follows:

$$\bigwedge_{q_1 \in Q, s \in \Sigma} \bigvee_{q_2 \in Q} \langle q_1, s, q_2\rangle_\delta \tag{1}$$

$$\wedge \bigwedge_{q_1, q_2 \in Q, q' \in Q', s \in \Sigma} \langle q_1, q'\rangle_G \wedge \langle q_1, s, q_2\rangle_\delta \Rightarrow \langle q_2, \delta'(q', s)\rangle_G \tag{2}$$

$$\wedge \bigwedge_{\substack{q_1, q_2, q_3 \in Q, q_1', q_2' \in Q', s \in \Sigma,\\ \delta'(q_2', s) \notin F', (q_1' \neq \delta(q_2', s)) \vee (q_1 \neq q_3)}} \begin{array}{r}\langle q_1, q_1', q_2, q_2'\rangle_N \wedge \langle q_2, s, q_3\rangle_\delta \\ \Rightarrow \quad \langle q_1, q_1', q_3, \delta'(q_2', s)\rangle_N\end{array} \tag{3}$$

$$\wedge \bigwedge_{\substack{q_1, q_2 \in Q, q_1', q_2' \in Q', s \in \Sigma,\\ q_1' \notin F', q_1' = \delta(q_2', s))}} \langle q_1, q_1', q_2, q_2'\rangle_N \wedge \langle q_2, s, q_1\rangle_\delta \Rightarrow \neg\langle q_1\rangle_F \tag{4}$$

$$\wedge \bigwedge_{\substack{q_1, q_2, q_3 \in Q, q_1', q_2' \in Q', s \in \Sigma,\\ q_1' \in F', q_1' \neq \delta(q_2', s)) \vee q_1 \neq q_3}} \begin{array}{r}\langle q_1, q_1', q_2, q_2'\rangle_A \wedge \langle q_2, s, q_3\rangle_\delta \wedge \\ \neg\langle q_3\rangle_F \Rightarrow \langle q_1, q_1', q_3, \delta'(q_2', s)\rangle_A\end{array} \tag{5}$$

$$\wedge \bigwedge_{\substack{q_1, q_2 \in Q, q_1', q_2' \in Q', s \in \Sigma,\\ q_1' \in F', q_1' = \delta(q_2', s))}} \langle q_1, q_1', q_2, q_2'\rangle_A \wedge \langle q_2, s, q_1\rangle_\delta \Rightarrow \langle q_1\rangle_F \tag{6}$$

$$\wedge \langle q_0, q_0' \rangle_G \wedge \bigwedge_{q \in Q, q' \in Q} (\langle q, q' \rangle_G \Rightarrow \langle q, q', q, q' \rangle_N) \wedge (\langle q, q' \rangle_G \Rightarrow \langle q, q', q, q' \rangle_A) \quad (7)$$

The variables $\langle \cdot \rangle_F$ represent whether a state is accepting. The transition function of the candidate automaton is defined in the variables $\langle \cdot \rangle_\delta$. The clauses (1) make sure that the transition function is total. For the vertices in G that are reachable from (q_0, q_0'), the first part of (7) and (2) enforce that the respective variables in $\langle \cdot \rangle_G$ are set to **true**. In particular, the first conjunct of (7) makes sure that (q_0, q_0') is defined as being reachable and (2) forces successors of reachable vertices in G to be marked as being reachable as well.

If there is path from some reachable vertex $(q_1, q_1') \in V$ to some vertex $(q_2, q_2') \in V$ in G such that no accepting state of $\mathcal{A}'$ is visited along the path, then $\langle q_1, q_1', q_2, q_2' \rangle_N$ indicates this fact. This is made sure by (3) in conjunction with (7). We use these path witness variables for detecting the loops in G that are non-accepting for $\mathcal{A}'$. The clauses (4) state that the states of $\mathcal{A}$ along such loops then also have to be non-accepting.

Dually, $\langle q_1, q_1', q_2, q_2' \rangle_A$ is set to true if there exists a path from some reachable vertex $(q_1, q_1') \in V$ to some vertex $(q_2, q_2') \in V$ in G such that no accepting state of $\mathcal{A}$ is visited in between. This is assured by the clauses (5) and (7). We add the clauses (6) to make sure that such a path may not form a loop if one of its $\mathcal{A}'$-states is accepting.

Note that there are no clauses enforcing that not too many variables in $\langle \cdot \rangle_G$, $\langle \cdot \rangle_\delta$, $\langle \cdot \rangle_N$ or $\langle \cdot \rangle_A$ are set to **true**. This is not necessary, as this only makes finding a satisfying assignment harder, but never results in false-positives. If a variable valuation satisfying all constraints has some state for which there is more than one transition possible for some input symbol, then the encoding makes sure that picking any of the possible successors always results in a correct DBA. Apart from the clauses (1), all conjuncts are Horn clauses (if we negate all values of $\langle \cdot \rangle_F$). While the generated instance is relatively large (of size $O(|Q'|^4)$), the fact that most clauses are of Horn type simplifies solving such SAT instances.

For speeding up the SAT solving process, symmetry breaking clauses [16] can also be added. For simplicity, we only break symmetry partially in our experimental evaluation. In particular, for $Q = \{q_1, \ldots, q_{|Q|}\}$ and $\Sigma = \{s_1, \ldots, s_{|\Sigma|}\}$, we add the following conjuncts that encode some relaxed form of lexicographical minimality of the candidate automaton over the automata whose graphs are isomorphic to the candidate solution:

$$\bigwedge_{1 \leq i < |Q|,\, i+1 < j \leq |Q|,\, (i-1) \cdot |Q| + j + 2 \leq k \leq |\Sigma|} \neg \langle q_i, s_k, q_j \rangle_\delta$$

4 Experimental evaluation[1]

We ran a prototype implementation of our technique on a couple of LTL formulas that are typical for model checking and synthesis. The upper part of Table

[1] Details and a downloadable implementation of the approach can be found at `http://react.cs.uni-saarland.de/tools/dbaminimizer`.

1 contains results for the 8 out of 12 LTL formulas stated in [7] that are representable as DBAs. In the lower half of the table, we give results for some typical synthesis specification parts and added some more complex formulas to allow for a more meaningful evaluation of our approach. Note that in both cases, the formulas occurring are mostly rather small, as in practice larger specifications are usually split up such that their conjuncts can be translated separately and composed to an overall automaton afterwards.

We used the tool `ltl2dstar v.0.5.1` [10] in conjunction with `ltl2ba v.1.1` [9] to obtain initial non-optimised deterministic Rabin automata equivalent to the input formulas given in the first column of the table. By applying the algorithm described in [11], we converted the deterministic Rabin automaton to a deterministic Büchi automaton whenever this is possible (and aborted otherwise). The `ltl2dstar` tool applies bisimulation quotienting, so the automata obtained are already heuristically optimised.

The number of states of these automata is given in the second column of Table 1. Columns four and five state the sizes of the reduction problems of these initial DBAs. The reduction process is repeated until no further improvements are possible. The resulting number of states in the minimised DBAs is shown in the third column. Finally, the total computation time of the SAT solver `picosat v.913` [3] observed on a computer with an Intel Core 2 Duo 1.86GHz processor for all reduction steps is given in the last column. We restricted the SAT solving time to one hour. Exceeding of this time bound is denoted by a star. Consequently, in such cases, the resulting DBA is not guaranteed to be minimal. The computation times for obtaining the initial DBA are negligible (< 0.05 seconds in all cases) and thus not added to the total time value. Furthermore, the computation of the SAT instances from the automata has also not been taken into account as most time was spent on writing the SAT instance to disk here, which can be circumvented by a future tighter integration with the SAT solver.

The table shows that except for one instance, the minimisation problem was always solved quickly. Thus, our technique is well-suited for being used as an optimisation step for the applications discussed in this paper. As the problem definition inherently induces a lot of symmetries in the SAT instance, we conjecture that future advancements in dynamic symmetry breaking [16] will allow tackling even bigger problem instances.

References

1. Baier, C., Katoen, J.P.: Principles of Model Checking. MIT Press (2008)
2. Barrett, C., Sebastiani, R., Seshia, S., Tinelli, C.: Satisfiability Modulo Theories. [5] 825–885
3. Biere, A.: Picosat essentials. JSAT **4**(2-4) (2008) 75–97
4. Biere, A.: Bounded Model Checking. [5] 457–474
5. Biere, A., Heule, M., van Maaren, H., Walsh, T., eds.: Handbook of Satisfiability. IOS Press (2009)
6. Bloem, R., Galler, S., Jobstmann, B., Piterman, N., Pnueli, A., Weiglhofer, M.: Specify, compile, run: Hardware from PSL. Electr. Notes Theor. Comput. Sci. **190**(4) (2007) 3–16

Table 1. Experimental results of our DBA minimisation technique.

LTL specification	# States		First instance		Total time
	From	To	# Vars	# Clauses	
$\mathsf{F}(q \wedge \mathsf{X}(p\mathsf{U}r))$	3	3	94	697	0.01 s
$p\mathsf{U}q\mathsf{U}r \vee q\mathsf{U}r\mathsf{U}p \vee r\mathsf{U}p\mathsf{U}q$	3	3	112	699	0.01 s
$\mathsf{F}(p \wedge \mathsf{X}(q \wedge \mathsf{XF}r))$	4	4	279	3785	0.01 s
$\mathsf{G}(p \rightarrow q\mathsf{U}r)$	5	3	852	13508	0.03 s
$p\mathsf{U}(q \wedge \mathsf{X}(r\mathsf{U}s))$	5	5	780	26972	0.04 s
$\mathsf{F}(p \wedge \mathsf{XF}(q \wedge \mathsf{XF}(r \wedge \mathsf{XF}s)))$	5	5	780	26972	0.01 s
$p\mathsf{U}(q \wedge \mathsf{X}(r \wedge \mathsf{F}(s \wedge \mathsf{XF}(u \wedge \mathsf{XF}(v \wedge \mathsf{XF}w)))))$	9	9	14752	5383278	30.36 s
$\mathsf{GF}p \wedge \mathsf{GF}q \wedge \mathsf{GF}r \wedge \mathsf{GF}s \wedge \mathsf{GF}u$	14	6	51467	13856026	63.03 s
$\mathsf{GF}a \vee \mathsf{GF}b \vee \mathsf{GF}c$	2	2	19	60	0.01 s
$\mathsf{G}(a \rightarrow \mathsf{F}b)$	4	2	339	1907	0.03 s
$\mathsf{G}(a\mathsf{U}b\mathsf{U}\neg a\mathsf{U}\neg b)$	4	2	339	1907	0.03 s
$(\mathsf{G}a \rightarrow \mathsf{F}b) \wedge (\mathsf{G}\neg a \rightarrow \mathsf{F}\neg b)$	4	4	291	1904	0.01 s
$\mathsf{G}\neg c \wedge \mathsf{G}(a \rightarrow \mathsf{F}b) \wedge \mathsf{G}(b \rightarrow \mathsf{F}c)$	5	2	852	13508	0.04 s
$\mathsf{G}(a \rightarrow \mathsf{F}b) \wedge \mathsf{G}c$	5	3	852	13508	0.03 s
$\mathsf{GF}(a \rightarrow \mathsf{XXX}b)$	7	2	3720	43457	0.08 s
$\mathsf{G}(a \rightarrow \mathsf{F}b) \wedge \mathsf{G}(\neg a \rightarrow \mathsf{F}\neg b)$	8	4	5635	89511	0.14 s
$\mathsf{GF}(a \leftrightarrow \mathsf{XX}b)$	9	6	8760	168358	1.57 s
$\mathsf{G}(a \rightarrow \mathsf{F}b) \wedge \mathsf{G}(b \rightarrow \mathsf{F}c)$	10	5	14247	589925	0.98 s
$\mathsf{G}(a \rightarrow \mathsf{XXX}b)$	10	9	16623	295076	3.71 s
$\mathsf{G}(a \rightarrow \mathsf{F}b) \wedge \mathsf{G}(c \rightarrow \mathsf{F}d)$	15	6	72660	9926065	23.96 s
$\mathsf{GF}(a \leftrightarrow \mathsf{XXX}b)$	17	15	116912	4752970	3617.3 s*

7. Etessami, K., Holzmann, G.J.: Optimizing Büchi automata. In Palamidessi, C., ed.: CONCUR. Volume 1877 of LNCS., Springer (2000) 153–167
8. Etessami, K., Wilke, T., Schuller, R.A.: Fair simulation relations, parity games, and state space reduction for Büchi automata. SIAM J. Comput. **34**(5) (2005) 1159–1175
9. Gastin, P., Oddoux, D.: Fast LTL to Büchi automata translation. In Berry, G., Comon, H., Finkel, A., eds.: CAV. Volume 2102 of LNCS. (2001) 53–65
10. Klein, J., Baier, C.: Experiments with deterministic ω-automata for formulas of linear temporal logic. Theor. Comput. Sci. **363**(2) (2006) 182–195
11. Krishnan, S.C., Puri, A., Brayton, R.K.: Deterministic w automata vis-a-vis deterministic buchi automata. In Du, D.Z., Zhang, X.S., eds.: ISAAC. Volume 834 of LNCS., Springer (1994) 378–386
12. Kupferman, O., Morgenstern, G., Murano, A.: Typeness for omega-regular automata. Int. J. Found. Comput. Sci. **17**(4) (2006) 869–884
13. Löding, C.: Efficient minimization of deterministic weak omega-automata. Inf. Process. Lett. **79**(3) (2001) 105–109
14. Piterman, N., Pnueli, A., Sa'ar, Y.: Synthesis of reactive(1) designs. In Emerson, E.A., Namjoshi, K.S., eds.: VMCAI, Springer (2006) 364–380
15. Pnueli, A., Rosner, R.: On the synthesis of a reactive module. In: POPL. (1989) 179–190
16. Sakallah, K.A.: Symmetry and Satisfiability. [5] 289–338

A Appendix

An example of an automatically minimised automaton: The following example shows that a SAT-based approach to the full minimisation of deterministic Büchi automata can reduce the size of the automata significantly. Taking the LTL formula $\psi = \mathsf{GF}(a \rightarrow \mathsf{XXX}b)$, the `ltl2dstar` tool computes an equivalent deterministic Rabin automaton with 7 states from it, which can easily be converted (automatically) to an equivalent DBA. The resulting DBA is in fact unnecessarily large, as the only way not to satisfy ψ on a word over the alphabet $2^{\{a,b\}}$ is to have a word ending with $\{a\}^\omega$. Consequently, there is an equivalent 2-state DBA which is found by our implementation of the technique described in this paper and which we depict in Figure 2.

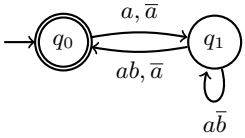

Fig. 2. A minimal DBA equivalent to the LTL formula $\psi = \mathsf{GF}(a \rightarrow \mathsf{XXX}b)$ over the atomic proposition set $\mathsf{AP} = \{a, b\}$. Accepting states are doubly-circled.

About the title of the paper: We used the word "precisely" in the title of the paper because some authors of other papers dealing with automaton size reduction have used the term "minimisation" for techniques that do not necessarily yield automata of minimal size (but rather use some rules to make them smaller in many cases). The word "precisely" was thus added to avoid confusion in this context.

Total transition functions in DBA: In this paper, we only considered deterministic Büchi automata with a *total* transition function. Whether this restriction is necessary depends on the application the DBA is supposed to be used in. For example, in the GR(1) synthesis approach [14], this is a necessary restriction whereas in probabilistic model checking, it is sometimes not necessary. In all cases, the minimisation of DBAs with a total transition relation is the same as for those with non-total transition relation: if the totality of the transition relation is not needed, the minimal automaton for the total-relation case can be made a minimal automaton for the non-total case by removing the (possibly existing) rejecting absorbing state from the automaton and removing all transitions to it.

NP-completeness of the problem considered: At the time of publication of this paper, whether the minimisation of deterministic Büchi automata is NP-complete was unknown. Recently, the NP-completeness has been proven by Schewe [19].

The applicability of our techniques to synthesising finite state systems: In the main part of the paper, we state that in generalized reactivity(1) synthesis (the more modern synthesis approach mentioned), all properties have to be representable and stated as DBA. While the original paper about this approach [14] does not state this fact, it has been clarified, for example, in [17].

The automata produced by LTL2DSTAR: The tool LTL2DSTAR does not necessarily produce Büchi automata whenever possible. Instead, it outputs automata having the more general Rabin or parity acceptance condition types (see, e.g., [18] for an overview). However, it has been shown [11, 12] that if there exists an equivalent DBA, then there is one that has the same set of states and the same transition relation. Thus, we only need to check if there is a suitable set of accepting states to obtain a DBA with the same language. This is however a simple task. We only need to find all non-accepting loops in the automaton, make the states on it non-accepting and mark the remaining states as being accepting. The result is then checked for equivalence with the original Rabin or parity automaton. If this is not the case, there exists no DBA for the specification given. This whole operation can be done time polynomial in the automaton size.

Why we used Picosat & only basic symmetry breaking: We tried a couple of SAT solvers for the experimental evaluation (Minisat, Picosat, and Precosat, the winner of 2009 SAT competition's application track) but found that Precosat usually performed worse on our examples than Picosat and Minisat spends too much time on preprocessing the instance such that it is usually slower than Picosat.

We also tried if Minisat or Precosat were able to solve the one SAT instance that remained unsolved in our experimental evaluation. Precosat was the only solver that was able to solve it in a reasonable amount of time (about 5 hours) and was only able to do so when we applied some more exhaustive symmetry breaking.

Nevertheless, five hours is currently too much for the formal methods community to call a technique as presented in this paper applicable. Thus, we left out results for the solvers other than Picosat and also didn't go into depth on more exhaustive symmetry breaking in the paper. As the results of Picosat are already good enough to show the main claim of this paper, namely that the Büchi automaton minimisation technique is applicable with today's solvers, we found this to be a reasonable choice.

More detailed experimental results: For completeness, we state the whole experimental results here. Due to the large number of instances, we splitted the overview into Table 2 and Table 3. Some of the instances were solved too quickly for the GNU/Linux `time` utility to be able to measure the computation time. For the table in the main part of the paper, we thus assumed a computation time of $0.01s$ in such cases. The two tables also contain some formulas we left out in the main part of the paper due to space restrictions. In the second column, it is always the number of states of the reference automaton that is given. Thus, for

example, a row with 5 states corresponds to a size reduction problem from 5 to 4 states.

References

17. Könighofer, R., Hofferek, G., Bloem, R.: Debugging formal specifications using simple counterstrategies. In: FMCAD, IEEE (2009) 152–159
18. Grädel, E., Thomas, W., Wilke, T., eds.: Automata, Logics, and Infinite Games: A Guide to Current Research. Volume 2500 of LNCS. Springer (2002)
19. Schewe, S.: Minimisation of deterministic parity and Büchi automata and relative minimisation of deterministic finite automata, 2010; arXiv/CoRR:1007.1333.

LTL specification	# states	Picosat		
		# Vars	# Clauses	Time (s)
$\mathsf{XX}a$	5	856	3408	0.00
$\mathsf{GF}(a \rightarrow \mathsf{XXX}b)$	7	3720	43457	0.03
	6	1935	18682	0.01
	5	888	6776	0.00
	4	339	1907	0.00
	3	96	355	0.00
	2	15	32	0.00
$\mathsf{F}(p \wedge \mathsf{XF}(q \wedge \mathsf{XF}(r \wedge \mathsf{XF}s)))$	5	780	26972	0.01
$\mathsf{F}(q \wedge \mathsf{X}(p\mathsf{U}r))$	3	94	697	0.00
$\mathsf{F}(p \wedge \mathsf{X}(q \wedge \mathsf{XF}r))$	4	279	3785	0.00
$p\mathsf{U}(q \wedge \mathsf{X}(r\mathsf{U}s))$	5	780	26972	0.04
$\mathsf{G}(a \rightarrow \mathsf{F}b) \wedge \mathsf{G}(c \rightarrow \mathsf{F}d)$	15	72660	9926065	8.62
	14	51311	6928183	5.54
	13	43032	4702964	3.85
	12	27423	3090265	2.51
	11	21080	1953967	1.40
	10	13995	1179668	1.00
	9	9528	673030	0.58
	8	5327	357709	0.24
	7	3270	173543	0.13
	6	1695	74527	0.09
$\mathsf{GF}a \wedge \mathsf{GF}b$	5	688	6768	0.00
	4	291	1904	0.00
	3	78	353	0.00
$\mathsf{GF}a \vee \mathsf{GF}b \vee \mathsf{GF}c$	2	19	60	0.00
$\mathsf{GF}a$	2	13	18	0.00
$a\mathsf{U}b\mathsf{U}c\mathsf{U}d$	5	780	26972	0.01
$\mathsf{G}(a \rightarrow \mathsf{F}b) \wedge \mathsf{G}c$	5	852	13508	0.01
	4	327	3788	0.00
	3	94	697	0.00
$(\mathsf{G}a \rightarrow \mathsf{F}b) \wedge (\mathsf{G}\neg a \rightarrow \mathsf{F}\neg b)$	4	291	1904	0.01
$p\mathsf{U}(q \wedge \mathsf{X}(r \wedge \mathsf{F}(s \wedge \mathsf{XF}(u \wedge \mathsf{XF}(v \wedge \mathsf{XF}w)))))$	9	14752	5383278	30.36
$\mathsf{G}(a \rightarrow \mathsf{F}b) \wedge \mathsf{G}(b \rightarrow \mathsf{F}c)$	10	14247	589925	0.45
	9	9016	336582	0.27
	8	5831	178915	0.14
	7	3276	86813	0.06
	6	1675	37292	0.04
	5	752	13504	0.02
$\mathsf{G}(a \rightarrow \mathsf{F}b) \wedge \mathsf{G}(\neg a \rightarrow \mathsf{F}\neg b)$	8	5635	89511	0.07
	7	3426	43451	0.04
	6	1575	18672	0.01
	5	788	6772	0.00
	4	291	1904	0.00

Table 2. Overview table of the complete results, part one. For each instance, we state the sizes of the SAT instances for each reduction step along with the SAT solving times.

LTL specification	# states	Picosat		
		# Vars	# Clauses	Time (s)
$\mathsf{GF}p \wedge \mathsf{GF}q \wedge \mathsf{GF}r \wedge \mathsf{GF}s \wedge \mathsf{GF}u$	14	51467	13856026	9.81
	13	35196	9405560	8.86
	12	24607	6180264	4.66
	11	19050	3907697	2.71
	10	14391	2359163	1.88
	9	9256	1345910	0.94
	8	5663	715318	0.58
	7	3552	347009	0.34
	6	1915	149002	33.25
$\mathsf{GF}(a \leftrightarrow \mathsf{XXX}b)$	17	116912	4752970	7.02
	16	85635	3470897	10.28
	15	60858	2481775	timeout
$\mathsf{G}(p \rightarrow q\mathsf{U}r)$	5	852	13508	0.01
	4	327	3788	0.00
	3	94	697	0.00
$\mathsf{GF}(a \leftrightarrow \mathsf{XX}b)$	9	8760	168358	0.17
	8	5187	89504	0.15
	7	2544	43433	0.07
	6	1395	18667	1.18
$\mathsf{G}\neg c \wedge \mathsf{G}(a \rightarrow \mathsf{F}b) \wedge \mathsf{G}(b \rightarrow \mathsf{F}c)$	5	852	13508	0.01
	4	327	3788	0.00
	3	112	699	0.00
	2	19	60	0.00
$\mathsf{G}(a \rightarrow \mathsf{XXX}b)$	10	16623	295076	0.29
	9	10704	168382	3.42
$\mathsf{G}(a \rightarrow \mathsf{F}b)$	4	339	1907	0.00
	3	96	355	0.00
	2	15	32	0.00
$\mathsf{G}(a\mathsf{U}b\mathsf{U}\neg a\mathsf{U}\neg b)$	4	339	1907	0.00
	3	96	355	0.00
	2	15	32	0.00
$p\mathsf{U}q\mathsf{U}r \vee q\mathsf{U}r\mathsf{U}p \vee r\mathsf{U}p\mathsf{U}q$	3	112	699	0.01

Table 3. Overview table of the complete results, part two. For each instance, we state the sizes of the SAT instances for each reduction step along with the SAT solving times.