Symbolic Implementation of Alternating Automata*

R. Bloem[1], A. Cimatti[2], I. Pill[1], M. Roveri[2], and S. Semprini[2]

[1] Graz University of Technology, {rbloem, ipill}@ist.tugraz.at
[2] ITC-irst, {cimatti, roveri, semprini}@itc.it

Abstract. We show how to convert alternating Büchi automata to symbolic structures, using a variant of Miyano and Hayashi's construction. We avoid building the nondeterministic equivalent of the alternating automaton, thus save an exponential factor in space.
For one-weak automata, Miyano and Hayashi's approach produces automata that are larger than needed. We show a hybrid approach that produces a smaller nondeterministic automaton if part of the alternating automaton is one weak.
We perform a thorough experimental analysis and conclude that the symbolic approach outperforms the explicit one.

1 Introduction

In this paper we consider two closely related problems: that of model checking specifications given as alternating Büchi automata (ABWs) and that of computing language emptiness for such automata. These problems have gained importance through the advent of new temporal logics such as PSL [1] and ForSpec [2].

The standard approach for model checking and consistency checking of Linear Time Logic (LTL) properties is to convert these properties to nondeterministic Büchi word automata (NBWs). This can be done explicitly [3–5] or symbolically [6]. LTL properties can be seen as one-weak alternating automata, a restrictive subclass of ABWs. A one-weak alternating automaton with n states can be translated to an NBW with $n \cdot 2^n$ states [5].

Languages like PSL are not star free and are therefore translated to ABWs, not to one-weak automata [7]. For ABWs in general, conversion to an NBW is not as simple as for one-weak automata. For ABWs, we need Miyano and Hayashi's construction [8], which generates $O(3^n)$ states [9]. Thus, an efficient implementation of this construction is the key to a successful application of alternating automata and, therefore, of logics

* Supported by the European Commission under contract 507219 (PROSYD).

like PSL. (It should be noted that most model checkers for PSL currently accept only a subset of the language.)

This paper presents two contributions to an efficient use of Miyano and Hayashi's construction. First, through a reformulation of the construction we are able to develop a symbolic approach. Since the size of the symbolic representation is polynomial in the size of the ABW, we can avoid the $O(3^n)$ blowup associated with the conversion to an NBW[3]. The resulting symbolic representation can be used for model checking and consistency checking using either BDDs or SAT.

The second contribution is a combination of Miyano and Hayashi's construction with that of Gastin and Oddoux for one-weak automata [5]. Thus, we are able to retain full generality while increasing efficiency for automata with parts that are one-weak. Such automata occur frequently and benefit significantly from our approach.

We perform a thorough experimental evaluation of our solution using both the classic explicit approach and the symbolic one using either BDDs or SAT-based model checking.

An approach through the linear μ-calculus [10] would also be possible. However, μ-calculus formulae are expressed using parity instead of Büchi automata. This makes computing emptiness in the unbounded case (not handled in [10]) much harder.

2 Preliminaries

We denote by $\mathcal{B}(V)$ the set of Boolean formulae with variables in V. Formulae in $\mathcal{B}^+(V)$ do not use negation, formulas in $\mathcal{B}^\vee(V)$ use only disjunction. *Valuations* are denoted by subsets of V. We use $\varphi[X/Y]$ to denote that each variable in Y is replaced by the corresponding variable or expression in X. We will assume a finite set of (Boolean) atomic propositions AP. Our alphabet Σ is 2^{AP}. The i-th letter of an infinite word w on alphabet Σ, where $w \in \Sigma^\omega$, is denoted by w_i, whereby the index starts at zero. The empty word is denoted by ϵ. A Σ-labeled tree τ is a prefix-closed set $T \subseteq \mathbb{N}^*$ together with a labeling function $L : T \to \Sigma$.

Definition 1. *An* alternating generalized Büchi word automaton (AGW) *is a tuple* $A = (Q, q_0, AP, \rho, Acc)$ *where* Q *is a finite nonempty set of states,* $q_0 \in Q$ *is the initial state,* AP *is a finite set of atomic propositions,* ρ *maps every state to a disjunction of formulae* $(\varphi \wedge \psi)$

[3] This blowup can also be avoided by an on-the-fly approach, but that is hard to combine with symbolic model checking.

where $\varphi \in \mathcal{B}(AP)$*, and* $\psi \in \mathcal{B}^+(Q)$ *(the automaton is* nondeterministic *if* $\psi \in \mathcal{B}^\vee(Q)$*), and* $Acc \subseteq 2^Q$ *is the acceptance condition. (The automaton is* non-generalized *if* $|Acc| = 1$*, in which case we will take Acc to be a subset of* Q*.)*

We abbreviate alternating/nondeterministic generalized/non-generalized Büchi word automaton to (A/N)(G/B)W. For nondeterministic automata, we will also write $\rho : Q \times 2^{AP} \to 2^Q$, as usual.

For a given $q \in Q$, $\rho(q)$ consists of a disjunctively related set of transition formulae $\varphi \wedge \psi$, where φ defines the set of labels for which the transition is valid and ψ defines the states to which the automaton will move.

A *run* of A on an infinite word $w \in (2^{AP})^\omega$ is a (possibly infinite) Q-labeled tree τ such that the label of the root (ϵ) is q_0 and for every node t labeled q, with $|t| = i$, the conjunction of w_i and the labels of t's children models $\rho(q)$. A branch may be finite if it ends in a node t labeled q with $|t| = i$ and w_i models $\rho(q)$. A run τ on w is *accepting* if every *infinite* branch has infinitely many labels in F for all $F \in Acc$. We denote by $\mathrm{L}(A)$ the set of words w for which A has an accepting run.

Note that in the definition of a run there is no requirement that the set of children of t be minimal. This condition can be added without changing the language. Note also that t can have an arbitrary, possibly empty set of children if $\rho(t) = \text{true}$. There is no run that contains a node labeled q if $\rho(q) = \text{false}$. A run of an NGW for word w can be viewed as sequence $r \in Q^\omega$.

A Büchi automaton induces a graph. The states of the automaton are the nodes of the graph and there is an edge from q to q' if q' occurs in $\rho(q)$. The graph is partitioned into maximal strongly connected components (SCCs), some of which may be trivial. A non-generalized alternating Büchi automaton is *weak* if each SCC contains either only accepting states or only non-accepting ones. The automaton is one weak (a.k.a. very weak or linear weak) if every SCC has size one.

Definition 2. *A* fair transition system (FTS) *[11] is a tuple* (V, A, T, Θ, F)*, where* V *is a finite set of* state variables*,* A *is a finite set of* input variables*,* $T \in \mathcal{B}(V \cup A \cup V')$ *is the* transition relation*,* $\Theta \in \mathcal{B}(V)$ *specifies a single* initial state*, and* $F \subseteq \mathcal{B}(V)$ *specifies the* acceptance condition*.*

In this definition, V', the set of primed versions of variables in V, is used to denote the next state variables.

An FTS $S = (V, A, T, \Theta, \{F_1, \ldots, F_n\})$ defines an NGW N as follows: $N = (Q, I, A, \rho, Acc)$, where Q is 2^V, I is defined by Θ, $\rho(q) = \bigvee\{\varphi \wedge \psi \mid \varphi \in \mathcal{B}(A), \psi \in \mathcal{B}^+(Q)$ and $q \models \varphi \wedge \psi \rightarrow T\}$, and $Acc = \{Acc_1, \ldots, Acc_n\}$, where $Acc_i = \{q \mid q \models F_i\}$. Thus, we can speak of a run of an FTS and the language of an FTS as if it were an NGW.

Where convenient, we will use the obvious extension to FTSs with variables with larger finite domains.

3 Converting the Alternating Automaton

In this section, we show how to construct an NGW from an ABW. Our construction combines the full generality of Miyano and Hayashi's approach with the efficiency of Gastin and Oddoux' approach for one-weak automata where possible. Since Miyano and Hayashi's original formulation refers to individual states and can not directly be encoded symbolically, we use an alternative formulation here, more closely related to that of [12]. We also show how the same approach can be used to convert an ABW to an FTS directly, avoiding building the NGW.

3.1 From ABW to NGW

Let $A = (Q, q_0, AP, \rho, Acc)$ be an ABW. Let $Q_S \subseteq Q$ be the set of states in SCCs of size greater than one and let $Q_W = Q \setminus Q_S$ be the set of states that are SCCs of size one. We partition Q into four sets: $Q_{SN} = Q_S \setminus Acc$, $Q_{SA} = Q_S \cap Acc$, $Q_{WN} = \{q \in Q_W \setminus Acc \mid$ SCC of q has a self loop$\}$, and $Q_{WA} = Q_W \setminus Q_{WN}$. Assume that $k = |Q_{WN}|$ and $Q_{WN} = \{b_1, \ldots, b_k\}$.

Theorem 1. *For the ABW $A = (Q, q_0, AP, \rho, Acc)$ there exists an NGW $A' = (Q', q'_0, AP, \rho', Acc')$ such that $\mathrm{L}(A') = \mathrm{L}(A)$. The NGW A' can be constructed as follows:*

- *Let $Q' = \mathcal{L} \times \mathcal{R} \times C$, with $\mathcal{L} = 2^Q$, $\mathcal{R} = 2^{Q_{SN}}$, and $C = \{0, \ldots, k\}$*
- *$q'_0 = (\{q_0\}, \emptyset, 0)$*
- *$Acc' = \{\mathcal{L} \times \{\emptyset\} \times C, \mathcal{L} \times \mathcal{R} \times \{1\}, \ldots, \mathcal{L} \times \mathcal{R} \times \{k\}\}$*
- *The transition relation ρ' is such that $(L', R', c') \in \rho'((L, R, c), \sigma)$, where $\sigma \in 2^{AP}$ and $L = \{l_1, \ldots, l_n\}$, if $\exists L'_1, \ldots, L'_n$ such that:*
  - *$\forall j : L'_j \cup \sigma \models \rho(l_j)$*
  - *$\bigcup_j L'_j = L'$*
  - *$R' \subseteq L'$*
  - *either $R = \emptyset$ and $R' = L' \cap Q_{SN}$, or $R \neq \emptyset$ and $R' \cup (L' \setminus Q_{SN}) \cup \sigma \models \bigwedge_{r \in R} \rho(r)$*

- *for all j, either $b_c \neq l_j$ or $b_c \notin L'_j$.*

Proof. A directed acyclic graph (DAG) can be converted to a tree in the obvious way. Thus, a DAG with labels in Q can be seen as a run.

It is well known that we can convert run trees to DAGs in such a way that an accepting run yields an accepting DAG and a non-accepting run yields a non-accepting DAG. The conversion is as follows. Suppose we have a run tree τ. On every level, the DAG will have at most one node with a given label. Thus for every level and a given label, we pick one representative node in τ. Other nodes with the same label are removed, as are their subtrees. Their incoming edges are redirected to the corresponding representative nodes. In the following we will consider runs as DAGs and identify nodes for a given level by their labels.

First, we will prove that $L(A) \subseteq L(A')$. Let DAG d be an accepting run of ABW A on word w. We construct the run r of NBW A' as follows. Let $r = r_0 r_1 \dots$ with $r_i = (L(i), R(i), c(i))$. Let $L(i) = \{q \mid q \text{ occurs at level } i \text{ of } d\}$. Assume $L(i) = \{l_1, \dots, l_n\}$ for some n and let $L'_i = \{q \mid q \text{ is a child of } l_i\}$. Then, by the definition of a run, we have $L'_i \cup w_i \models \rho(l_i)$ and $L(i+1) = \bigcup_i L'_i$.

Let $R(0) = \emptyset$, let $R(i+1) = L(i+1) \cap Q_{SN}$ if $R(i) = \emptyset$, and let $R(i+1) = \{q \in L(i+1) \mid q \in Q_{SN}, \exists q' \in R(i) : q \text{ is a child of } q' \text{ in } d\}$ if $R(i) \neq \emptyset$.

Finally, choose $c(0) = 0$ and choose $c(i)$ such that there is no edge from $b_{c(i)}$ at level i to $b_{c(i)}$ at level $i+1$ and the level j such that $j < i$ and $c(j) = c(i)$ is minimal. (States in Q_{WN} are numbered $b_1, \dots, b_k$.)

It should be clear that r satisfies the definition of the transition relation of A' and is therefore a run. Choose levels d_i in d such that $d_0 = 0$ and for all i, d_{i+1} is the level closest to the root such that an accepting state occurs on every path between level d_i and level d_{i+1}. Since d is an accepting run, there are infinitely many such levels, and R is empty at those levels. Furthermore, since no path gets stuck in a state b_j, for any b_j there are infinitely many levels at which the edge from b_j to b_j is not taken, and thus $c(i)$ is equal to j for infinitely many values of i. Therefore, r is accepting.

Vice-versa, we need to prove that $L(A') \subseteq L(A)$. Let $r = r_0 r_1 \dots$ be an accepting run of A' on w, with $r_i = (L(i), R(i), c(i))$. We will construct an accepting run DAG d.

By the definition of the NBW, we have

$$\forall q \in L(i) \;\; \exists L' \subseteq L(i+1) : Q'_{I_i} \cup w_i \models \rho(q), \text{ and} \tag{1}$$

$$\forall q \in R(i) \;\; R' = R(i+1) \cup (L(i+1) \setminus Q_{SN}) \Rightarrow R' \cup w_i \models \rho(q). \tag{2}$$

Let $S(i)$ denote the set of states on level i of d. We will define $S(i)$ inductively. Let $S(0) = \{q_0\}$. We define $S(i+1)$ as follows. For $q \in S(i) \setminus R(i)$, we pick as successors the states $L' \subseteq L(i+1)$ defined by 1. For $q \in S(i) \cap R(i)$ we pick as successors the states $R' \subseteq L(i+1) \cup R(i+1)$ as defined in 2. (Note that L' and R' are not uniquely defined. Any sets satisfying 1 and 2 will do.)

Note that $R(i+1) \subseteq R' \subseteq S(i+1) \subseteq L(i+1) \cup R(i+1)$. The last inclusion may be strict if $S(i) \subseteq R(i)$ and $L(i+1)$ contains redundant states in Q_{SN}.

By 1 and 2, d is a run.

Since for all i the successors of a state in $R(i)$ are all either in $R(i+1)$ or outside Q_{SN}, all paths between two levels i and j with $R(i) = R(j) = \emptyset$ must contain at least one state outside Q_{SN}.

Now, since for every j and for infinitely many i, we have $c(i) = j$, the transition from b_j to b_j is avoided infinitely often. Thus, no path in d gets stuck in any state b_j. Since every path has infinitely many states outside Q_{SN}, and does not get stuck in a state $b_j \in Q_{WN}$, it must visit an accepting state infinitely often, and the DAG is accepting. □

Intuitively, if we take $Q_{WA} \cup Q_{WN} = \emptyset$, then the construction reduces to a variant of Miyano and Hayashi's approach. A run τ of A can be mapped to a run r' of A' such that if (L, R, c) is the state of r' after i transitions, then (1) L is the set of labels of the nodes on level i of τ and (2) R consists of all labels in L that label a state v for which there is no accepting state on the path between the last level with $R = \emptyset$ and v. As Miyano and Hayashi note, a run of an ABW is accepting if and only if it has infinitely many levels such that each path between two such levels contains an accepting state. This is the case if and only if R becomes empty infinitely often.

On the other hand, if $Q_{WA} \cup Q_{WN} = Q$, then the construction reduces to that of Gastin and Oddoux. Every state in Q_{WN} has a number i, and if the self loop on state i is taken, then $c \neq i$.

Using our combined approach the state space of A' has size $|Q_{WN}| \cdot 3^{|Q \setminus (Acc \cup Q_{WN})|} \cdot 2^{|Acc \cup Q_{WN}|}$ versus $3^{|Q \setminus Acc|} \cdot 2^{|Acc|}$ using Miyano and Hayashi's approach.

Note that there is no requirement in the transition relation that the sets L' and R' be minimal. This condition can be added, but is hard to deal with in a symbolic implementation. We do use the minimality condition when constructing A' explicitly.

We refer to our construction as **MHGO**, as it combines Miyano and Hayashi's with Gastin and Oddoux' construction. By setting $Q_{WN} =$

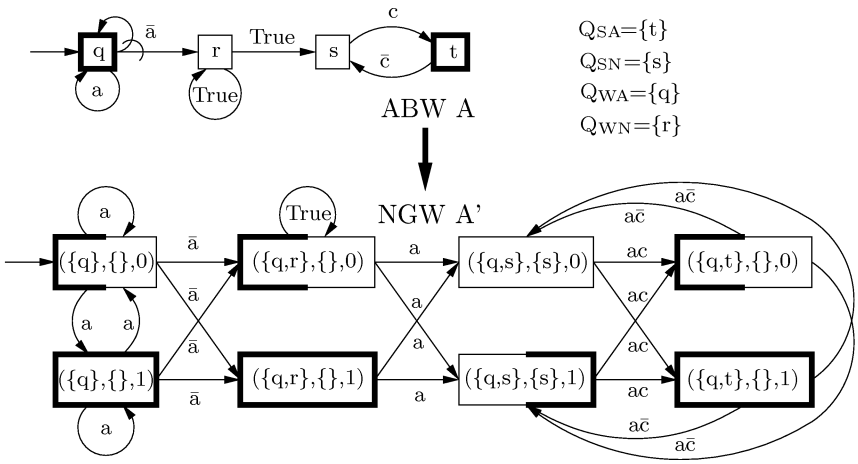


Fig. 1. ABW A and part of corresponding NGW A'.

$Q_{WA} = \emptyset$, our construction reduces to that of Miyano and Hayashi and we refer to it as **MH**.

Using this construction, accepting loops have length at least k. To remedy this drawback we introduce a simple **variant of MHGO** in which C has domain $2^{Q_{WN}}$. Intuitively a $c \in C$ contains all the states on which a self loop is not allowed, and each state in Q_{WN} should appear infinitely often. Although this expands the state space, its symbolic encoding is not larger.

The example shown in Fig. 1 illustrates the construction. (We use concatenation to denote conjunction and ā (a bar) to indicate negation.) Since A' is very large, we only show a part: for most states edges labeled with ā are missing. In the upper half of A' we have that $c = 0$ and in the lower one that $c = 1$. From the illustration we see that the transitions of the two halves are almost symmetric, but for the second column of states. Since r is in Q_{WN} we can take the self-loop on r only when c is 0. The NGW has two acceptance sets. One requiring infinitely many visits in states with $R = \emptyset$, the other infinitely many visits in states with $c = 1$, meaning that the self loop on r is not taken. In Fig. 1, the first acceptance set is illustrated by bold borders on the left, the second by bold borders on the right of the state.

3.2 From ABW to Fair Transition System

Let $A = (Q, q_0, AP, \rho, Acc)$ be an ABW. Let Q_{SN}, Q_{SA}, Q_{WN}, and Q_{WA} be as before. We define the symbolic fair transition system S to be (V, AP, T, Θ, F), where

- $V = Q_L \cup Q_R \cup \{c\}$ where $Q_L = \{q_L \mid q \in Q\}$, $Q_R = \{q_R \mid q \in Q_{SN}\}$, and c has domain $\{0, \ldots, k\}$,
- $T = T_I \wedge T_{LC} \wedge T_R$ (the parts are defined below),
- $\Theta = q_{0L} \wedge (c = 0)$, and
- $F = (F_0, \ldots, F_k)$, with $F_0 = \bigwedge_{q \in Q_{SN}} \neg q_R$ and $F_i = (c = i)$ for $1 \leq i \leq k$.

Let $[Q'_{LR}/Q]$ be the substitution that replaces $q \in Q \setminus Q_{SN}$ with q'_L and $q \in Q_{SN}$ with q'_R. We have $T_I = \bigwedge_{q \in Q_{SN}} (q'_R \rightarrow q'_L)$,

$$T_{LC} = \bigwedge_{q \in Q \setminus Q_{WN}} (q_L \rightarrow \rho(q)[Q'_L/Q]) \wedge \bigwedge_{q = b_i \in Q_{WN}} (q_L \rightarrow \rho(q)[Q'_L/(Q \setminus \{q\}) \text{ and } (c \neq i \wedge q'_L)/\{q\}]),$$

$$T_R = (F_0 \wedge \bigwedge_{q \in Q_{SN}} (q'_L \rightarrow q'_R)) \vee (\neg F_0 \wedge \bigwedge_{q \in Q_{SN}} (q_R \rightarrow \rho(q)[Q'_{LR}/Q])).$$

The FTS is easily seen to encode the NGW A' introduced in the last section. A valuation v corresponds to a state q of the NGW: $q = (L, R, \hat{c})$, where $L = \{q \mid q_L \in v\}$, $R = \{q \mid q_R \in v\}$, and $\hat{c}$ is the valuation of variable c. Thus, by Theorem 1, we have the following theorem.

Theorem 2. *$L(S) = L(A)$.*

The size of the FTS depends significantly on that of ρ. In contrast to Miyano and Hayashi's approach, where T contains two copies of ρ for $Q \setminus Acc$ and one for $Q \cap Acc$, the combined approach needs only one copy for non-accepting states that are either trivial SCCs or in Q_{WN}. On the other hand, it contains additional propositions $c \neq i$ for every self-loop on a state in Q_{WN} and it has multiple fairness conditions.

Using the simple variant of MHGO in which C has domain $2^{Q_{WN}}$, it is possible to find witnesses in fewer steps than with MH, because we do not require minimality for T. For the PSL formula $\mathsf{always}\{\{a; b\}[*n]\}$! we can find a witness in one step, whereas MH needs $2n$ steps.

4 Experimental evaluation

We implemented the approaches on top of the NuSMV model checker [13]. We compare the **explicit** approach of building the NGW first and then converting it to an FTS against our direct **symbolic** approach of building the FTS. We use both **BDD**s and **SBMC** [14]. Finally, we compare the **MH** approach with the **variant of MHGO**. It turns out that the

variant of MHGO sometimes outperforms MHGO, especially when the language of the automaton is not empty. The reason is that the latter creates accepting loops that are longer than necessary. We use the pattern $\{\mathrm{E},\mathrm{S}\}_{\{\mathrm{MH},\mathrm{MHGO}\}}$-$\{\mathrm{BDD},\mathrm{SBMC}\}$ to denote the combinations of encodings and engines. (Where MHGO refers to the variant.) All experiments were run on a 3GHz Intel Xeon CPU with 4GB of memory, with a time out of 900s and a 1GB memory limit.

In the SAT approach we use Simple Bounded Model Checking (SBMC) since it is complete and allows for a fair comparison with BDDs, and we used MiniSAT [15] as SAT engine. The variable order chosen for the BDD experiments is such that the current and the corresponding next variables are consecutive and each q_R immediately follows the corresponding q_L. (This ordering yields good performance on average.) We first compute the set of reachable states and use it in the language emptiness algorithms to restrict the search as is common practice in model checking. Using this setting we obtained better results on average.

We experimented with two classes of ABWs: R-ABWs are random ABWs for which the number of accepting states, labels variables, transitions, and destinations of each transition are proportional to the number of states; and PSL-ABWs, which are built from typical PSL expressions used in industry [16].

The results of the experimental analysis are reported in Fig. 2: we plot the number of problems solved in a given amount of time (the samples are ordered by increasing computation time). The results show that the symbolic encoding outperforms the explicit one, and that the best approach is either S_{MH}-BDD or S_{MH}-SBMC. This is due to the construction of the explicit NBW which caused all the time-outs/memory-outs that occurred for E_{MH}-BDD on R-ABWs.

S_{MH}-SBMC outperforms S_{MH}-BDD on ABWs with $L(A) \neq \emptyset$ because of the limited number of steps needed by S_{MH}-SBMC to find a solution. (See Figs. 2(a) and 2(c).) On the other hand, BDDs perform better than SBMC on ABWs with $L(A) = \emptyset$ (Figs. 2(b) and 2(d)): SBMC needs to consider a high depth to be able to conclude that the language is empty, which results in a high consumption of resources. (Cf. [14]). Note that in these figures SBMC is either very fast or times out. A typical example of a property with an easy induction proof is $\mathsf{G}(p \wedge \mathsf{X}^n(\neg p \wedge \mathsf{X}\,\varphi))$. A typical property for which the induction proof is hard is $\mathsf{G}(p \wedge \mathsf{F}(\neg p \wedge \mathsf{X}\,\varphi))$. We conjecture that this is related to the fact that the encoding does not require minimality and thus the induction proof depends on φ.

Figs. 2(e) and 2(f) compare symbolic MH with MHGO on random PSL properties. No significant difference results on these formulae. In the unsatisfiable case, on average we noticed that S_{MHGO}-SBMC times out at a lower depth than S_{MH}-SBMC. The presence of multiple fairness conditions appears to blow up the SAT instance that is generated to prove that there is no witness. In the satisfiable cases S_{MHGO}-BDD performs slightly worse than S_{MH}-BDD because of the increased number of fix-points needed to perform language emptiness. On the other hand, S_{MHGO}-SBMC performs slightly better since the SAT instances are smaller and shorter witnesses are found.

Figs. 2(g) and 2(h) show the results of computing language emptiness of the combination of the Gigamax model (from the NuSMV distribution) and R-ABWs with $L(A) = \emptyset$. The language of the combination is obviously empty. The automata used in the plots of Fig. 2(g) are those for which S_{MH}-SBMC was not able to prove language emptiness. In contrast, in Fig. 2(h) we use automata for which S_{MH}-SBMC suceeded. The plots show that the BDD-based approaches handle the increased complexity of the combined model well. SBMC, however shows a mixed picture. For S_{MH}-SBMC, Fig. 2(g) shows that there are three cases for which adding the Gigamax model enables language emptiness to complete. On the other hand, Fig. 2(h) shows that there are 15 examples for which S_{MH}-SBMC can not compute language emptiness on the combined model although it could do so on the automaton in separation. E_{MH}-SBMC can show language emptiness for 8 of the 20 automata used in Fig. 2(g), but only for three of the combined models. On the other hand, for the automata used in Fig. 2(h) the results for E_{MH}-SBMC are confirmed. With random models, too, BDDs are quite well behaved, but the performance of SBMC is hard to predict.

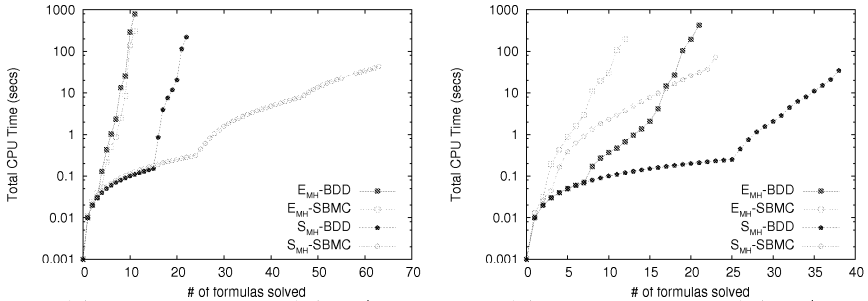
The experimental analysis clearly shows that the symbolic encoding outperforms the explicit encoding, and that on average SBMC is the most effective technique if the language is nonempty. On the other hand, BDDs are more effective than SBMC if the language is empty. Our results for SBMC confirm those of [14]: sometimes the search needs large resources to consider deep runs to prove that the property holds. (This is often the case when the language of the automaton is empty.) Finally, on the experiments considered there is no evident benefit in using the variant of MHGO over MH. We believe that a more thorough experimental analysis is needed to confirm or refute this result.

We are currently exploring the effects of optimizing both the ABW and the NBW using the techniques of [17, 18]. Preliminary results are

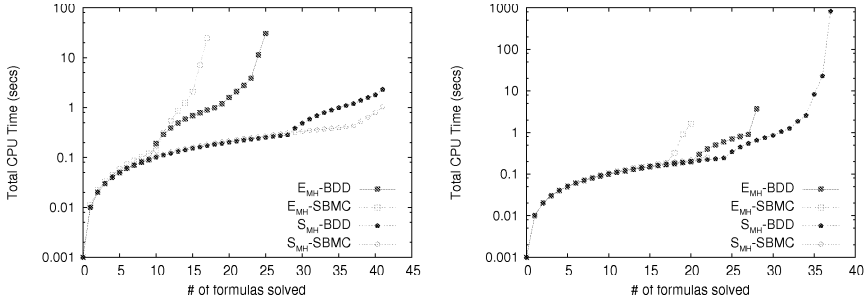
promising, but a thorough analysis must be carried out to better understand the impact of such optimizations. We are also researching ways to apply the optimizations of [17] directly to the symbolic encoding.

References

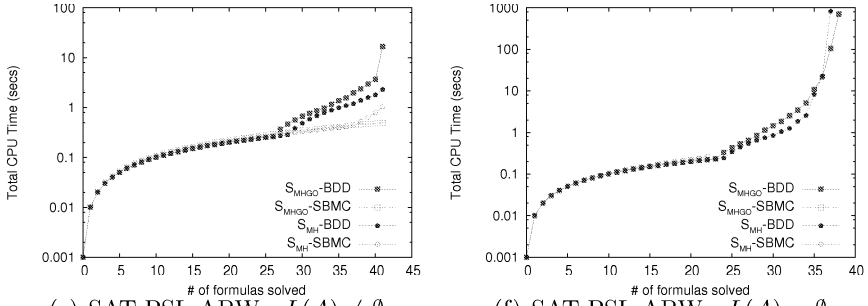
1. IEEE-Commission: IEEE standard for Property Specification Language (PSL) (2005) IEEE Std 1850-2005.
2. Armoni, R., et al.: The ForSpec temporal logic: A new temporal property-specification language. In: International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS'02). (2002) 296–311
3. Vardi, M.Y., Wolper, P.: An automata-theoretic approach to automatic program verification. In: Proc. Logic in Computer Science. (1986) 322–331
4. Somenzi, F., Bloem, R.: Efficient Büchi automata from LTL formulae. In: Twelfth Conference on Computer Aided Verification (CAV'00). (2000) 248–263
5. Gastin, P., Oddoux, D.: Fast LTL to Büchi automata translation. In: Conference on Computer Aided Verification (CAV '01). (2001) 53–65
6. Clarke, E.M., Grumberg, O., Hamaguchi, K.: Another look at LTL model checking. Formal Methods in System Design (1) (1997) 47–71
7. Ben-David, S., Bloem, R., Fisman, D., Griesmayer, A., Pill, I., Ruah, S.: Automata construction algorithms optimized for PSL. http://www.prosyd.org (2005) Prosyd D 3.2/4.
8. Miyano, S., Hayashi, T.: Alternating finite automata on ω-words. Theoretical Computer Science **32** (1984) 321–330
9. Leucker, M.: Logics for mazurkiewicz traces. Technical Report AIB-2002-10, RWTH, Aachen, Germany (2002)
10. Jehle, M., Johannsen, J., Lange, M., Rachinsky, N.: Bounded model checking for all regular properties. In: 3rd Workshop on Bounded Model Checking (BMC'05). Volume 144.1 of Electronic Notes in Theoretical Computer Science. (2005) 3–18
11. Manna, Z., Pnueli, A.: The Temporal Logic of Reactive and Concurrent Systems, Specification. Springer Verlag, New York (1992)
12. Kupferman, O., Vardi, M.Y.: Weak alternating automata are not that weak. ACM Transactions on Computational Logic **2**(3) (2001) 408–429
13. Cimatti, A., Clarke, E., Giunchiglia, F., Roveri, M.: NuSMV: a new Symbolic Model Verifier. In: Conference on Computer-Aided Verification. (1999) 495–499
14. Heljanko, K., Junttila, T.A., Latvala, T.: Incremental and complete bounded model checking for full PLTL. In: Computer Aided Verification (CAV'05). (2005) 98–111
15. Eén, N., Sörensson, N.: MiniSAT (2005) `http://www.cs.chalmers.se/Cs/Research/FormalMethods/MiniSat/Main.html`.
16. David, S.B., Orni, A.: Property-by-Example guide: a handbook of PSL/Sugar examples - PROSYD deliverable d1.1/3. http://www.prosyd.org (2005)
17. Fritz, C.: Constructing Büchi automata from linear temporal logic using simulation relations for alternating Büchi automata. In: Conference on Implementation and Application of Automata. (2003) 35–48 LNCS 2759.
18. Fritz, C., Wilke, T.: Simulation relations for alternating Büchi automata. Theoretical Computer Science **338** (2005) 275–314



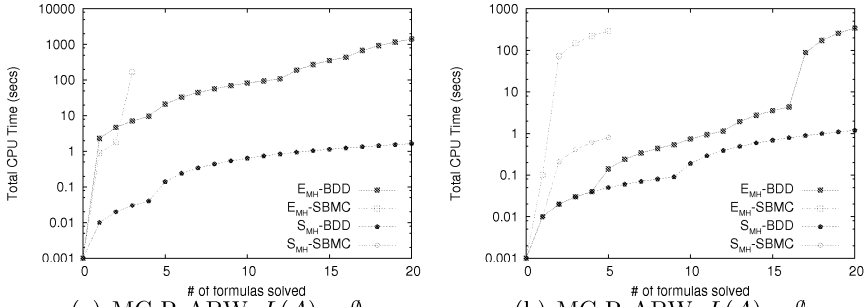
(a) SAT R-ABWs, $L(A) \neq \emptyset$ (b) SAT R-ABWs, $L(A) = \emptyset$



(c) SAT PSL-ABWs, $L(A) \neq \emptyset$ (d) SAT PSL-ABWs, $L(A) = \emptyset$



(e) SAT PSL-ABWs, $L(A) \neq \emptyset$ (f) SAT PSL-ABWs, $L(A) = \emptyset$



(g) MC R-ABW, $L(A) = \emptyset$ (h) MC R-ABW, $L(A) = \emptyset$

Fig. 2. Language emptiness and Model checking results.