Bounded Model Checking for Weak Alternating Büchi Automata

Keijo Heljanko[1,⋆], Tommi Junttila[1], Misa Keinänen[1,⋆⋆], Martin Lange[2], and Timo Latvala[3,⋆⋆⋆]

[1] Laboratory for Theoretical Computer Science
Helsinki University of Technology
P.O. Box 5400, FI-02015 TKK, Finland
{Keijo.Heljanko, Tommi.Junttila, Misa.Keinanen}@tkk.fi
[2] Institut für Informatik Ludwig-Maximilians-Universität München, Germany
Martin.Lange@ifi.lmu.de
[3] Department of Computer Science, University of Illinois at Urbana-Champaign, USA
tlatvala@uiuc.edu

Abstract. We present an incremental bounded model checking encoding into propositional satisfiability where the property specification is expressed as a weak alternating Büchi automaton (WABA). The encoding is linear in the specification, or, more exactly $O(|I| + k \cdot |T| + k \cdot |\delta|)$, where $|I|$ is the size of the initial state predicate, k is the bound, $|T|$ is the size of the transition relation, and $|\delta|$ is the size of the WABA transition relation. Minimal length counterexamples can also be found by increasing the encoding size to be quadratic in the number of states in the largest component of the WABA. The proposed encoding can be used to implement more efficient bounded model checking algorithms for ω-regular industrial specification languages such as Accellera's Property Specification Language (PSL). Encouraging experimental results on a prototype implementation are reported.
Keywords: Weak Alternating Büchi Automata, Bounded Model Checking, PSL, NuSMV

1 Introduction

Large and demanding verification efforts require that the property specification language used is up to the task. Linear temporal logic (LTL), the property specification language implemented in many model checkers, has been criticised for the lack of expressive power [1,2]. Expressing certain properties in LTL is cumbersome at best, and writing assumptions for compositional reasoning can even be impossible. Most of these shortcomings are in one way or another related to the fact that LTL cannot express all ω-regular languages. This has been recognised by many key players in the hardware industry and Accellera's Property Specification Language (PSL) [3,4] has been proposed

⋆ Supported by the Academy of Finland (projects 112016, 213113).
⋆⋆ Supported by the Academy of Finland (project 211025) and Helsinki Graduate School in Computer Science and Engineering (HeCSE).
⋆⋆⋆ Supported by the Academy of Finland (project 109539) and the Emil Aaltonen Foundation.

as a solution. PSL extends LTL in many ways, but perhaps most importantly PSL can express all ω-regular languages.[4]

Expressive specification languages require efficient model checking techniques to deliver on their promise. *Bounded model checking* (BMC) [5] is a symbolic model checking technique that focuses on searching for bounded counterexamples to the given property. By encoding the model checking problem to propositional satisfiability (SAT), bounded model checking can leverage the efficiency of modern SAT-solver technology. Encoding BMC to SAT is accomplished by writing a propositional formula that models all executions of the system of certain length. Additional constraints ensure that the final formula is satisfiable if some execution is a counterexample. There are also methods for concluding that current reached depth is enough to prove that the given property holds [6,7,8,9]. BMC has established itself as an important tool among current verification techniques. A very important question is therefore, can BMC *efficiently* model check *all* ω-regular properties, especially those expressed in PSL.

This work explores different possibilities of implementing BMC for PSL by using the automata theoretic approach to model checking. The PSL property can first be converted into an alternating Büchi automaton (ABA) with the help of an external translation procedure, such as the one described by the Prosyd project (see [10]). This procedure can create so called *weak alternating automata* (WABA) which have certain restrictions on the structure of the automaton but are still able to express all ω-regular properties. A large subset of core PSL can be converted into a WABA with a linear number of states with a few exceptions [10].

In the rest of the paper we explore different options of creating an efficient BMC encoding for WABAs. With an exponential blow-up ($O(2^a+3^b)$, where a is the number of accepting states and b is the number of non-accepting states) the WABA can be converted to an explicit state nondeterministic Büchi automaton using the Miyano-Hayashi construction [11]. This explicit state Büchi automaton could be used but the size of the encoding is in the worst case exponential in the size of the WABA.

A significantly better option would be to implement a symbolic version (SAT encoding) of the Miyano-Hayashi construction [10]. However, this approach does not exploit the weakness of the ABAs and might thus not be an optimal approach for WABAs. We have also experimentally observed that neither the symbolic nor the explicit state versions of the approach preserve minimal length counterexamples.

We present a new efficient BMC encoding specialised for model checking WABAs. The size of the encoding is *linear* in the specification as WABA and the system model. By increasing the size of the encoding to be quadratic in the number of states in the largest component of the WABA, we can guarantee that it detects minimal length counterexamples for all WABAs. The encoding utilises the incremental SAT encoding framework developed in [9].

We have experimentally evaluated our new BMC encoding for WABAs. Compared to BMC based on explicit state Büchi automata, the new WABA encoding is much more robust because the exponential blow-up in the explicit state Miyano-Hayashi construction is avoided. The new linear size encoding is clearly faster than a symbolic BMC

[4] PSL can also express properties of finite words, for simplicity only ω-words are considered here.

encoding of the Miyano-Hayashi construction. In addition, the minimal counterexample variant of our new encoding produces shorter counterexamples in some cases. On LTL formulas the new encoding generates minimum length counterexamples and is as compact (within a constant factor) as the most compact specialised LTL encodings known [9]. Furthermore, the performance on LTL is quite similar.

There is some earlier work on bounded model checking for subclasses of alternating Büchi automata and for all ω-regular properties. Sheridan [12] describes a *non-incremental* BMC encoding for very weak alternating Büchi automata. This encoding captures only the LTL subset of ω-regular properties since very weak alternating Büchi automata exactly correspond to LTL properties [13,14]. A BMC encoding for alternation-free μTL, a temporal logic that can express all ω-regular properties, has been developed by Jehle et al. [15]. The encoding is cubic in the used bound k and thus not as efficient as the new encoding presented in this work.

2 Alternating Büchi Automata

In this section we cover the technical definitions needed to introduce our BMC encoding for WABAs. The set of positive Boolean formulas over $\mathcal{X}$, denoted by $\mathcal{B}^+(\mathcal{X})$, is the smallest set of formulas which contains all elements from $\mathcal{X}$ and is closed under disjunction and conjunction. A subset S of $\mathcal{X}$ is a model of $\theta \in \mathcal{B}^+(\mathcal{X})$, denoted by $S \models \theta$, iff the truth assignment that assigns true to the elements of S and false to the elements of $\mathcal{X} \setminus S$ satisfies θ.

As alphabet Σ of alternating automata we restrict ourselves to only considering valuations of atomic propositions. More precisely, for a given non-empty finite set AP of atomic propositions we define the set of atomic proposition complements $\overline{AP} = \{\overline{p} \mid p \in AP\}$ and let Σ be the largest set $\Sigma \subseteq 2^{AP \cup \overline{AP}}$ such that for all $p \in AP$ exactly one element of $\{p, \overline{p}\}$ is contained in each member of Σ.

An alternating Büchi automaton (ABA) is of the form $A = (Q, \Sigma, q_0, \delta, F)$, where Q is a finite set of states, Σ is a finite alphabet, $q_0 \in Q$ is the initial state, $\delta : Q \to \mathcal{B}^+(AP \cup \overline{AP} \cup Q)$ is the transition relation and $F \subseteq Q$ is the set of accepting states. We use $\mathcal{B}^+(A)$ to denote the set of Boolean formulas that occur in A's transition function.

Given an infinite word $w \in \Sigma^\omega$, w_i denotes the i-th letter of w (i.e. $w = w_0 w_1 w_2 \ldots$). A run of $A = (Q, \Sigma, q_0, \delta, F)$ on w is a directed acyclic graph (dag) $G = (V, E)$ with the following properties:

- $V \subseteq Q \times \mathbb{N}$,
- $E \subseteq \bigcup_{i \geq 0}((Q \times \{i\}) \times (Q \times \{i+1\}))$,
- $(q_0, 0) \in V$,
- if $(q, i) \in V$ then $(w_i \cup \{q' \mid ((q, i), (q', i+1)) \in E\}) \models \delta(q)$, and
- if $((q, i), (q', i+1)) \in E$ then both $(q, i) \in V$ and $(q', i+1) \in V$.

For technical convenience this definition of a run allows for states which are unreachable from the initial state. Let σ be an infinite path in a run in G, i.e. an infinite sequence of nodes $(v_0, v_1, v_2, \ldots)$ such that $(v_i, v_{i+1}) \in E$ for all $i \geq 0$. Let $Inf(\sigma)$ be the set of states that consists of all automaton states appearing infinitely often in the nodes of σ.

An infinite path σ is accepting iff $F \cap Inf(\sigma) \neq \emptyset$. A run G is accepting iff every infinite path through G is accepting. An ABA $A = (Q, \Sigma, q_0, \delta, F)$ accepts a word $w \in \Sigma^\omega$ iff there is an accepting run G of the automaton A on w. The definition of a run allows a state to have no successors and a path through the run (as well as the whole run) to be finite. In effect all such finite paths ending in a state with no successors are "accepting". Alternatively the existence of states with no successors could be easily ruled out by placing additional constraints on $\delta(\cdot)$.

Example 1. For instance, $\delta(q_1) = ((p \wedge q_1) \vee (\overline{p} \wedge ((r \wedge (q_2 \wedge q_3)) \vee \overline{r})))$ means that from state $(q_1, i) \in V$ with valuation $w_i = \{p, r\}$ move to a state set at $i+1$ containing $\{q_1\}$ (this also happens with valuation $\{p, \overline{r}\}$), while with valuation $\{\overline{p}, r\}$ we will move to a state set containing $\{q_2, q_3\}$. With valuation $\{\overline{p}, \overline{r}\}$ the transition relation of q_1 becomes true, which means that we do not require q_0 to have any successors.

A weak alternating Büchi automaton (WABA) is an ABA $A = (Q, \Sigma, q_0, \delta, F)$ whose states Q can be partitioned into *components* $Q_1 \uplus \cdots \uplus Q_m$ such that:[5]

- for all $j, k \in \{1, \ldots, m\}$, $q_j \in Q_j$, $q_k \in Q_k$: if q_k appears syntactically in $\delta(q_j)$ then $k \leq j$; and
- for all $1 \leq j \leq m$: $Q_j \subseteq F$ or $Q_j \cap F = \emptyset$.

A WABA is a *very* weak alternating Büchi automaton (VWABA) if no component Q_j contains more than one state. For a component Q_j, $|\delta_j|$ denotes the sum of the sizes of the transition relations $\delta(q)$, where $q \in Q_j$.

Let A be a WABA with state set Q partitioned into components $Q_1 \uplus \cdots \uplus Q_m$ and final state set F. We next define the *component unrolling depth* d_j needed to detect minimal length counterexamples in our BMC encoding for each component Q_j. For any $j \in \{1, \ldots, m\}$ let

$$d_j \;=\; \begin{cases} 0 & , \text{ if } Q_j \subseteq F \\ |Q_j| & , \text{ if } Q_j \cap F = \emptyset \end{cases}$$

3 Incremental Bounded Model Checking for Weak Alternating Büchi Automata

Our incremental encoding for weak alternating automata is based on the simple BMC encodings [16,17,9] for LTL. The approach to incrementality used here is exactly the same as in [9]. First of all, the encoding needs to be formulated so that it is easy to derive the encoding for bound $k = i+1$ from the encoding for bound $k = i$. This is done by separating the encoding to a *k-invariant* part and a *k-dependent* part. The information learned by the SAT solver from the *k*-invariant constraints can be reused when the bound is increased while the *k*-dependent constraints and all the information learned from them needs to be discarded. Thus we try to minimise the use of *k*-dependent constraints in our

[5] Given an ABA the sets $Q_1, \ldots, Q_m$ can be easily computed by using an algorithm for computing the maximal strongly connected components (MSCCs) in a graph induced by the ABA transition relation as follows: the states are the nodes, and there is an edge from q_j to q_k iff q_k appears syntactically in $\delta(q_j)$.

encoding. The so called *Base constraints* are also k-invariant, but they are conditions that are constant for all $0 \leq i \leq k$.

As in earlier works, paths of length k are encoded using k-invariant *model constraints* $|[M]|_k$. They encode initialised finite paths of the model M of length k:

$$|[M]|_k \Leftrightarrow I(s_0) \wedge \bigwedge_{i=1}^{k} T(s_{i-1}, s_i),$$

where $I(s)$ is the initial state predicate and $T(s,s')$ is a total transition relation. Let $\pi = s_0 s_1 s_2 \ldots$ be an initialised infinite path through M. The corresponding word $w = w_0 w_1 w_2 \ldots \in \Sigma^\omega$ is obtained by concatenating the sets of valuations of atomic propositions in the states s_i. We say that π is a (k,l)-loop if $\pi = (s_0 s_1 \ldots s_{l-1})(s_l \ldots s_k)^\omega$ such that $0 < l \leq k$ and $s_{l-1} = s_k$.

The *loop constraints* also closely follow [9] by employing $k+1$ fresh *loop selector variables* $l_0, \ldots, l_k$. They constrain the finite path of the system to always be a (k,i)-loop for exactly one i, in which case the variable l_i is true and all other l_j variables are false. Many k-dependent constraints are avoided by introducing a new special system state s_E with fresh (unconstrained) state variables acting as a *proxy state* for the endpoint of the path. In the k-dependent part the proxy state s_E is constrained to be equivalent to s_k. The variable InLoop_i is true iff the state s_i belongs to the loop part of a (k,l)-loop. These are encoded by conjuncting the constraints below and denoted by $|[\mathit{LoopConstraints}]|_k$:

Base	$l_0 \Leftrightarrow \bot$
	$\text{InLoop}_0 \Leftrightarrow \bot$
$k-$invariant	$l_i \Rightarrow (s_{i-1} = s_E)$
$1 \leq i \leq k$	$\text{InLoop}_i \Leftrightarrow \text{InLoop}_{i-1} \vee l_i,$
	$\text{InLoop}_{i-1} \Rightarrow \neg l_i$
$k-$dependent	$\text{InLoop}_k \Leftrightarrow \top$
	$s_E \Leftrightarrow s_k$

We will first give an encoding that detects minimal length counterexamples for *all* WABAs, and later on show an optimisation that makes the encoding linear in the size of the WABA if this requirement is dropped. Given a WABA A, in our new encoding the state variables of the system are split at each time i to the actual state variables s_i of the system, to the set of variables for all automata states $|[s_q]|_i^d$ (one for $0 \leq i \leq k+1$ and each pair (q,d), where $q \in Q_j$ and $0 \leq d \leq d_j$). The encoding also contains a few additional variables which will be referred to explicitly. The rules of the encoding are given as a set of Boolean constraints.

The *WABA constraints* $|[A_{WABA}]|_k$ are new to this work and restrict the bounded paths defined by the model constraints and loop constraints to infinite words accepted by WABA A. One intuition for understanding the encoding is given by the fact that for (k,l)-loops the semantics of branching and linear time coincide. We will in fact employ algorithmic ideas similar to those used in branching time logic CTL model checkers.

The transition relation of A is encoded in a straightforward manner. For each component Q_j and for each state $q \in Q_j$ the following constraints are created:

	$0 \leq d \leq d_j$
Base	$\|[s_{q_0}]\|_0^0 \Leftrightarrow \top$, where q_0 is the initial state
$k-$invariant, $0 \leq i \leq k$	$\|[s_q]\|_i^d \Leftrightarrow \|[\delta(q)]\|_i^d$

where the $k-$invariant encoding $\|[\delta(q)]\|_i^d$ for each component Q_j, and for each state $q \in Q_j$ is the following:

$\|[\delta(q)]\|_i^d$	$0 \leq i \leq k, 0 \leq d \leq d_j$
$\|[p]\|_i^d$	$\|[p]\|_i^d \Leftrightarrow p_i$
$\|[\overline{p}]\|_i^d$	$\|[\overline{p}]\|_i^d \Leftrightarrow \neg p_i$
$\|[q']\|_i^d$	$\|[q']\|_i^d \Leftrightarrow \|[s_{q'}]\|_{i+1}^d$, if $q' \in Q_j$
	$\|[q']\|_i^d \Leftrightarrow \|[s_{q'}]\|_{i+1}^0$, if $q' \notin Q_j$
$\|[\psi_1 \wedge \psi_2]\|_i^d$	$\|[\psi_1 \wedge \psi_2]\|_i^d \Leftrightarrow \|[\psi_1]\|_i^d \wedge \|[\psi_2]\|_i^d$
$\|[\psi_1 \vee \psi_2]\|_i^d$	$\|[\psi_1 \vee \psi_2]\|_i^d \Leftrightarrow \|[\psi_1]\|_i^d \vee \|[\psi_2]\|_i^d$

In the encoding above p_i denotes the variable holding the value of the atomic proposition p in the state s_i. Notice how for state $q \in Q_j$ the successor states q' inside Q_j get the values from the current unrolling d while the successor states q' outside Q_j get their values from the unrolling $d = 0$. The intuition for this will be explained below.

We use a *proxy loop state* indexed with L with associated (free) automaton variables $\|[s_q]\|_L^d$ to act as the loop state in order to make as many constraints k-invariant as possible. For non-accepting components the k-dependent rules bind the truth values of $\|[s_q]\|_{k+1}^d$ to $\|[s_q]\|_L^{d+1}$ (jump to the next unrolling level $d+1$), while for accepting components they bind the values of $\|[s_q]\|_{k+1}^0$ to the value of $\|[s_q]\|_L^0$, i.e. to the values at the loop point state of the same unrolling. This is encoded by conjuncting the following constraints for each component Q_j and for each state $q \in Q_j$:

	$0 \leq d \leq d_j$
Base	$\|[s_q]\|_L^{d_j+1} \Leftrightarrow \bot$, if $q \notin F$
$k-$invariant, $1 \leq i \leq k$	$l_i \Rightarrow \left(\|[s_q]\|_L^d \Leftrightarrow \|[s_q]\|_i^d\right)$
$k-$dependent	$\|[s_q]\|_{k+1}^d \Leftrightarrow \|[s_q]\|_L^{d+1}$, if $q \notin F$
	$\|[s_q]\|_{k+1}^0 \Leftrightarrow \|[s_q]\|_L^0$, if $q \in F$

The intuitive idea behind the encoding is as follows. Our encoding can be seen as a SAT implementation of an automata theoretic *branching time* model checker using WABAs such as [18] but specialised for models induced by (k,l)-loops. Because of the component structure of the WABA, each component Q_j can assume that all other components and atomic propositions it refers to have already been evaluated, and the results are

available. This is all that is needed to evaluate the component Q_j by iteratively substituting these subresults.[6]

Similarly to [18] we want to compute the effect of these substitutions in terms of a fixpoint evaluation procedure. Consider a non-accepting component Q_j first. We want $|[s_q]|_L^1$ to evaluate to whether at the loop point L starting from a state $q \in Q_j$ the automaton has some run which accepts the ω-word induced by the loop. Because we do not want to allow accepting runs to be trapped forever in a non-accepting component, the fixpoint required is the least fixpoint, and gives us the initial approximation values $|[s_q]|_L^{d_j+1} \Leftrightarrow \bot$. By running through the loop once in the *backward direction* making substitutions of known results along the way, we can get a better approximation of the final value, namely $|[s_q]|_L^{d_j}$. Either we have already reached a fixpoint, or at least one of states $q' \in Q_j$ has obtained the value $|[s_{q'}]|_L^{d_j} = \top$, in which case we have to resubstitute this value by running through the loop a second time in the backward direction. Clearly after $d_j = |Q_j|$ rounds the fixpoint is guaranteed to be reached, and the values of $|[s_q]|_L^1$ are exact results of the fixpoint iteration. Finally, an extra fixpoint iteration is done with $|[s_q]|_i^0$ variables to get the correct final values for indices to the right of the loop point.

We could do the obvious dual greatest fixpoint iteration for the accepting components. However, we will use the optimisation trick of employing *any fixpoint* instead of the greatest fixpoint. The intuitive reason why this is sound is that any fixpoint will in our encoding cautiously underapproximate the greatest fixpoint, (see the soundness proof, Lemma 1 in Appendix A which never uses the fact that the fixpoint obtained for accepting components is the greatest fixpoint). The completeness part is trivial, as the any fixpoint enforcing constraints are strictly less constraining that the constraints that would be needed for enforcing the exact greatest fixpoint.

We can optionally add constraints based on the monotonicity of the fixpoint approximations of non-accepting components. These $k-$invariant propagation constraints are as follows. For each non-accepting component Q_j, and for each state $q \in Q_j$, $0 \leq i \leq k+1, 1 \leq d \leq d_j$:

$k-$invariant	$\|[s_q]\|_i^d \Rightarrow \|[s_q]\|_i^{d-1}$

Conjuncting all the constraints above the encoding $|[M,A_{WABA}]|_k$ becomes:

$$|[M,A_{WABA}]|_k \Leftrightarrow |[M]|_k \wedge |[\mathit{LoopConstraints}]|_k \wedge |[A_{WABA}]|_k.$$

Theorem 1. *Given a finite Kripke structure M and a WABA A, M has a path π accepted by A iff there exists a $k \in \mathbb{N}$ such that $|[M,A_{WABA}]|_k$ is satisfiable. More specifically, if $\pi = s_0 s_1 s_2 \ldots$ is a (k,l)-loop accepted by A then $|[M,A_{WABA}]|_k$ is satisfiable.* [7]

Proof. Immediate by Lemmas 1 and 2 in Appendix A. □

[6] Notice the similarity to evaluating CTL formulas by substituting subformula results and propagating these in the backward transition relation direction. See for example the WABA based CTL model checking algorithm [18] as well as similar algorithms for the alternation free μ-calculus [19]. The main difference is that we aim for an easy encoding into SAT instead of optimal running time as in the algorithms mentioned above.

[7] A direct corollary of this is that minimal length (k,l)-loop witnesses can be detected.

The exact size of the encoding is $O(|I|+k\cdot|T|+k\cdot|\delta|+k\cdot\sum_{j=1}^{m}(d_j\cdot|\delta_j|))$. Note that the size is bounded from above by $O(|I|+k\cdot|T|+k\cdot|Q|\cdot|\delta|)$, and becomes $O(|I|+k\cdot|T|+k\cdot|\delta|)$ when the WABA is a very weak alternating Büchi automaton (as produced by most LTL to WABA translations). Combined with a linear size translation from an LTL formula into a VWABA (for example a state acceptance based variant of [20] with a symbolically encoded transition relation), bounded LTL model checking using this approach is as compact as the approaches of [16,17,9]. In fact, by doing so the encoding would for LTL formulas effectively become an optimised incremental variant of [16].

Trading Minimal Length Witnesses for a Smaller Encoding. Instead of quantifying d over $0 \leq d \leq d_j$ in the encoding above, for any non-accepting component Q_j we can instead use $0 \leq d \leq c_j$, where $1 \leq c_j \leq d_j$. Now c_j is the number of fixpoint iterations made[8], and we need the following constraints to guarantee correctness of the approach. For each non-accepting component Q_j, and for each state $q \in Q_j$ the following fixpoint-enforcing constraints are added:

Base	$\lvert[s_q]\rvert_L^0 \Leftrightarrow \lvert[s_q]\rvert_L^1$

The constraints intuitively check that the fixpoint iteration has reached a fixpoint after c_j iterations. Thus the approach will be sound. The reason why the approach is still complete is that by going through the loop part of a (k,l)-loop d_j times one can with $c_j = 1$ simulate the d_j fixpoint iterations done by going through the loop part only once but with d_j unrollings. Thus increasing the bound by roughly a factor of d_j can compensate for the lack of d_j unrollings. By changing the quantification to, for example, always use $c_j = 1$ (as used in our experiments) the resulting encoding is of size $O(|I|+k\cdot|T|+k\cdot|\delta|)$, i.e. linear in the size of the WABA. The correctness of the encoding is preserved in the sense that every witness will eventually be detected when the bound is increased large enough (albeit with a non-minimal bound).

4 Experimental Results

We have implemented a prototype of the proposed WABA BMC encoding on top of a development version of the NuSMV tool [21]. We use the "Sugar" tool (obtained from `http://www.prosyd.org/`), by C. J. Kargl of TU Graz, as a translator from PSL to ABAs and reuse our previous incremental SAT encoding techniques [9]. As the SAT solver we use ZChaff version 2004.11.15 in the experiments. In order to evaluate and validate the proposed encoding, we have also implemented two other BMC approaches for WABAs on top of the same software platform: (i) translate the WABA to an explicit state Büchi automaton by using the Miyano-Hayashi algorithm of the "Sugar" tool and then do BMC by using the explicit state Büchi automaton, (ii) take the Miyano-Hayashi translation from (W)ABA to Büchi automata given in [10, page 38] and derive a *symbolic* BMC encoding from it.

[8] The encoding of $\lvert[s_q]\rvert_i^d$ with $d = 0$ can be seen as an "extra" fixpoint iteration. It is needed in order to also obtain correct $\lvert[s_q]\rvert_i^0$ values for indices i to the "right" of the loop point. We use it here to also check that the fixpoint has been reached.

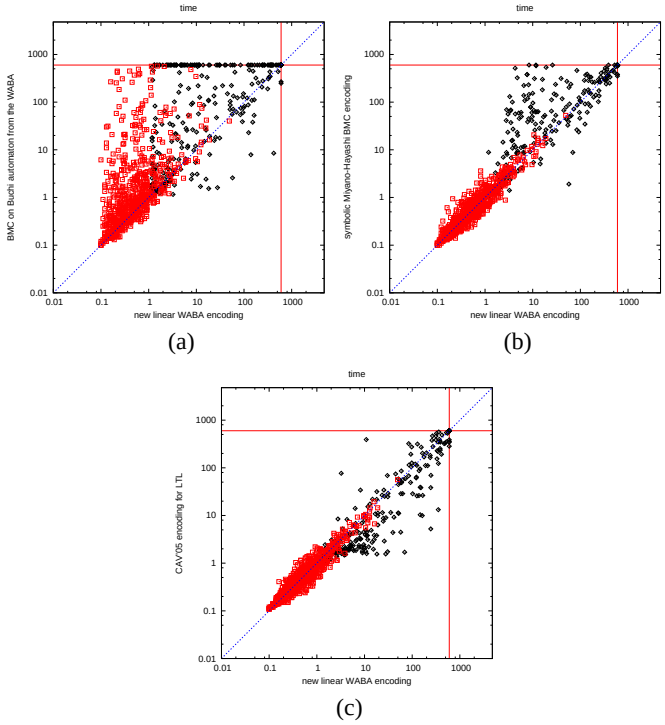

Fig. 1. A comparison of encoding approaches on random models and VWABAs generated from LTL formulae. Red boxes mark cases with a counterexample while black diamonds mark cases where none was found.

These two BMC encodings are linear in the bound k and the sizes of the transition relations of the corresponding automata (Büchi and WABA, resp.). Unfortunately we do not have space to explain them in more detail here. The prototype implementation as well as the experiments are available at `http://www.tcs.hut.fi/~timo/cav2006`. The implementation also contains a (W)ABA input path, allowing alternative PSL to (W)ABA translations to be used.

Figures 1 and 2 show a comparison of encoding schemes for randomly generated models (Kripke structures of 100 states and a single justice fairness requirement) and WABAs generated from LTL and PSL formulae (of parse tree sizes between 3 and 14). The time limit for each run was 10 minutes and the memory limit 1.5GiB.

In Fig. 1(a), 1(b) and 1(c), we benchmark our new algorithm on 1200 random LTL formulae. We plot the total execution time of each run to either find a counterexample for the property or to reach the bound limit of 50. In the plots, cases where a counterexample was found are denoted by red boxes while black diamonds denote cases where none of the approaches found a counterexample. The scales are logarithmic. Based on Fig. 1(a), it is easy to see that the "WABA to Büchi" approach is not very competitive: it suffers from the automata size blow-up occurring during the WABA to explicit state

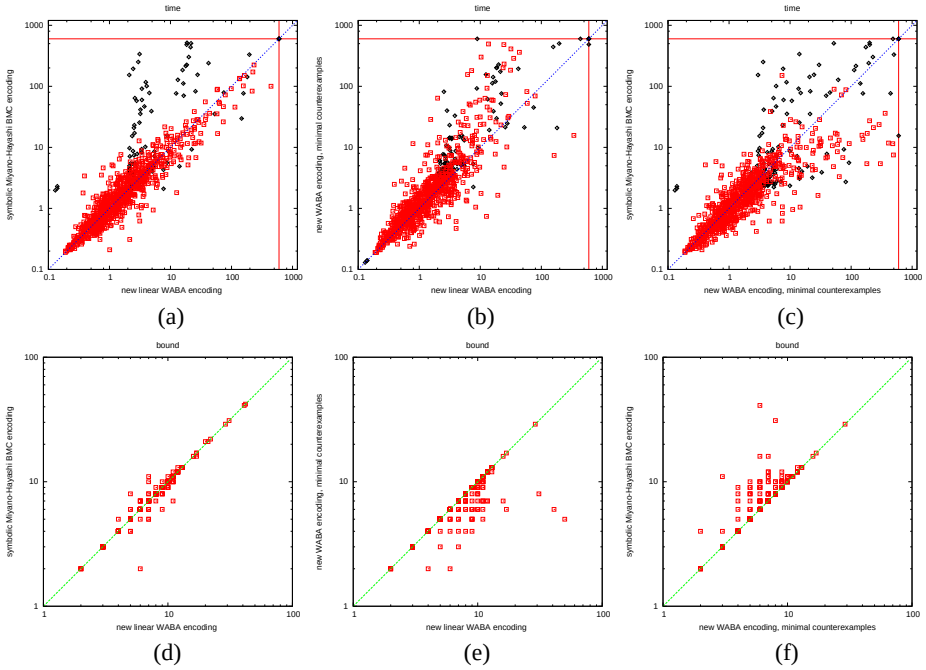

Fig. 2. A comparison of encoding approaches on random models and WABAs generated from PSL formulae. Red boxes mark cases with a counterexample while black diamonds mark cases where none was found.

Büchi automata translation. We can see that the proposed WABA BMC encoding is competitive against the symbolic Miyano-Hayashi approach (Fig. 1(b)). As expected, the specialised LTL encoding of [9] performs slightly better than the new, more general encoding but the difference is not large: the new encoding seems to be a reasonably good BMC algorithm for LTL, too.

In Fig. 2, we compare the encodings on 1000+ WABAs obtained by generating random PSL formulas, translating them to ABAs using the "Sugar" tool, and picking those instances which are WABAs that are *not* very weak. It is known that in the version of the "Sugar" tool used by us there are some discrepancies with respect to the semantics of PSL, but that does not effect our use of it as a random WABA generator. The bound and other parameters of the setup, as well as plot point encoding, are identical to the LTL case. We also plot the *bound reached*, i.e. the counterexample length, for the runs that found one. The scales are logarithmic.

The new linear encoding performs better than the symbolic Miyano-Hayashi encoding, as shown in Fig. 2(a), with comparable counterexample lengths, as can be observed from Fig. 2(d). Comparing the two new encodings in Fig. 2(b), the linear encoding is clearly faster but may generate significantly longer counterexamples as shown in Fig. 2(e). If we were to model check systems with a larger transition relation, the increased counterexample length as seen here might sometimes translate into a slower

running time. Comparing the new encoding that can find minimal counterexamples to the symbolic Miyano-Hayashi encoding in Figures 2(c) and 2(f) we see that there is no clear winner in speed but that the new encoding produces shorter counterexamples.

To sum up, these results show that the proposed WABA BMC is a competitive encoding for WABAs generated from PSL formulas, and quite close to a state-of-the-art BMC encoding specialised for LTL.

5 Conclusions

Our new BMC encoding for WABAs seems very competitive. With BMC using explicit state Büchi automata, it is obvious that for complicated properties the potentially exponential conversion from a WABA will become a bottleneck. The reason why our encoding performs better than a symbolic Miyano-Hayashi encoding is not completely clear to us. We speculate that the more deterministic nature of our encoding generates easier problems for the SAT solver. The fact that the new encoding can exploit the structure of WABAs unlike Miyano-Hayashi, which works for all alternating automata, may also help. Both are linear size in the specification, but if we use a version that is in the worst case quadratic in the number of states in the largest component of the WABA, our new encoding is guaranteed to find minimal length counterexamples.

The proposed WABA BMC encoding can be made complete (in the sense that it can also prove properties, not only find counterexamples) by modifying and applying the simple-path constrains of [9] in a straightforward way.

We would like to investigate whether it is possible to modify Miyano-Hayashi to generate tight Büchi automata. We believe that the BMC encoding of this work can be adapted to also generate a symbolic WABA to Büchi automaton conversion procedure (an alternative to Miyano-Hayashi for WABAs) which generates tight Büchi automata and thus detects minimal length counterexamples along the lines of [22]. This intuition is based on the fact that [22] is an adaptation of the PLTL BMC encoding [17] to the symbolic Büchi automaton setting and the implementation techniques used here are quite similar to those of [17].

Other potential future directions of research are related to succinctness. One possibility would be to devise new direct BMC encodings for general, non-weak ABAs or for alternating parity automata. Generalising the encoding to temporal logics with past operators (e.g. PSL extended with past) may potentially involve handling of two-way alternating automata.

Acknowledgements The authors would like to thank I. Niemelä and H. Tauriainen for interesting discussions and pointers on the topic. Thanks also to R. Bloem and other contributors of the Prosyd project for their freely available PSL translation tool as well as A. Cimatti, M. Roveri, and the rest of the NuSMV team for assistance and providing us with a development version of NuSMV.

References

1. Lichtenstein, O., Pnueli, A., Zuck, L.D.: The glory of the past. In: Logic of Programs. Volume 193 of LNCS., Springer (1985) 196–218

2. Vardi, M.: Branching vs. linear time: Final showdown. In: TACAS. Volume 2031 of LNCS., Springer (2001) 1–22
3. Accellera: Property specification language: Reference manual – version 1.1 (2004) `http://www.eda.org/vfv/docs/PSL-v1.1.pdf`.
4. IEEE: IEEE Standard 1850 - Property Specification Language (PSL) (2005)
5. Biere, A., Cimatti, A., Clarke, E., Zhu, Y.: Symbolic model checking without BDDs. In: TACAS. Volume 1579 of LNCS., Springer (1999) 193–207
6. Sheeran, M., Singh, S., Stålmarck, G.: Checking safety properties using induction and a SAT-solver. In: FMCAD. Volume 1954 of LNCS., Springer (2000) 108–125
7. McMillan, K.L.: Interpolation and SAT-based model checking. In: CAV. Volume 2725 of LNCS., Springer (2003) 1–13
8. Awedh, M., Somenzi, F.: Proving more properties with bounded model checking. In: CAV. Volume 3114 of LNCS. (2004) 96–108
9. Heljanko, K., Junttila, T., Latvala, T.: Incremental and complete bounded model checking for full PLTL. In: CAV. Volume 3576 of LNCS., Springer (2005) 98–111
10. Ben-David, S., Bloem, R., Fisman, D., Griesmayer, A., Pill, I., Ruah, S.: Automata constructon algorithms optimized for PSL. Technical Report Deliverable 3.2/4, ProdSyd project (2005) Available from: `http://www.prosyd.org/`.
11. Miyano, S., Hayashi, T.: Alternating finite automata on ω-words. Theoretical Computer Science **32** (1984) 321–330
12. Sheridan, D.: Bounded model checking with SNF, alternating automata, and Büchi automata. Electronic Notes in Theoretical Computer Science **119** (2005) 83–101
13. Rohde, G.S.: Alternating Automata and the Temporal Logic of Ordinals. PhD thesis, University of Illinois at Urbana-Champaign (1997)
14. Löding, C., Thomas, W.: Alternating automata and logics over infinite words. In: IFIP TCS2000. Volume 1872 of LNCS., Springer (2000) 521–535
15. Jehle, M., Johannsen, J., Lange, M., Rachinsky, N.: Bounded model checking for all regular properties. Electronic Notes in Theoretical Computer Science **144** (2006) 3–18
16. Latvala, T., Biere, A., Heljanko, K., Junttila, T.: Simple bounded LTL model checking. In: FMCAD. Volume 3312 of LNCS., Springer (2004) 186–200
17. Latvala, T., Biere, A., Heljanko, K., Junttila, T.: Simple is better: Efficient bounded model checking for past LTL. In: VMCAI. Volume 3385 of LNCS., Springer (2005) 380–395
18. Kupferman, O., Vardi, M.Y., Wolper, P.: An automata-theoretic approach to branching-time model checking. J. ACM **47** (2000) 312–360
19. Cleaveland, R., Steffen, B.: A linear-time model-checking algorithm for the alternation-free modal mu-calculus. Formal Methods in System Design **2** (1993) 121–147
20. Gastin, P., Oddoux, D.: Fast LTL to Büchi automata translation. In Berry, G., Comon, H., Finkel, A., eds.: CAV. Volume 2102 of LNCS., Springer (2001) 53–65
21. Cimatti, A., Clarke, E.M., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., Tacchella, A.: NuSMV 2: An OpenSource tool for symbolic model checking. In: CAV. Volume 2404 of LNCS., Springer (2002) 359–364
22. Schuppan, V., Biere, A.: Shortest counterexamples for symbolic model checking of LTL with past. In: TACAS. Volume 3440 of LNCS., Springer (2005) 493–509

Appendix A - Proofs

Here we prove the soundness and completeness of the encoding.

Lemma 1. *Given a finite Kripke structure M, a WABA A and a $k \in \mathbb{N}$, if $||[M, A_{WABA}]||_k$ is satisfiable then there is an initialised infinite path π through M such that the induced word w is accepted by A.*

Proof. Suppose $|[M, A_{WABA}]|_k$ has a satisfying truth assignment β for its variables. Since β satisfies $|[M]|_k$ there are states $s_0 s_1 \dots s_k$ that form an initialised finite path in M. Note that $|[\mathit{LoopConstraints}]|_k$ requires that there is $0 < l \leq k$ such that $s_k = s_{l-1}$. Let π now be the initialised infinite path $s_0 \dots s_{l-1}(s_l \dots s_k)^\omega$ through M. It remains to be seen that the corresponding word w is accepted by A.

We will prove the following stronger statement from which the claim of the theorem follows because of the base constraint for the initial state q_0. For a word $w = w_0 w_1 w_2 \dots \in \Sigma^\omega$ let $w^{(i)}$ denote the suffix of w starting from w_i. We use A_q to denote the WABA that results from A by making q the initial state. *For all components Q_j of A, all $0 \leq d \leq d_j$, all $q \in Q_j$, and all $0 \leq i \leq k$: if $\beta(|[s_q]|_i^d) = \top$ then $w^{(i)}$ is accepted by A_q.*

Note that the topological order on A's components is well-founded. Hence, we can use Noetherian induction assuming that the statement has been proved for all lower components already.

Let Q_j be a final component. Take any $q \in Q_j$ and assume $\beta(|[s_q]|_i^0) = \top$ for some $0 \leq i \leq k$. It is straightforward to construct a run dag for A_q and $w^{(i)}$ starting with the node (q,i). The constraints for δ then require $\beta(|[\delta(q)]|_i^0) = \top$.[9] Since Boolean connectives in δ are uniformly translated in the constraints for δ, there must be a model Q' of $\delta(q)$. The construction of the run dag is then iterated on the next level with nodes $(q', i+1)$ for some $q' \in Q'$. Note that the constraints always ensure that there are models of $\delta(q)$ for each q that occurs in this construction. This continues on each infinite path of the run ad infinitum or until a state q' is reached such that $q' \notin Q_j$. But then, by weakness, q' must belong to some component for which an accepting run dag has already been constructed by the induction hypothesis. Note that all the states on such infinite paths that remain in component Q_j are final. Hence, the run dag is accepting, and we have $w^{(i)}$ is accepted by A_q.

Now let Q_j be a non-final component. Again, take any $q \in Q_j$ but now assume $\beta(|[s_q]|_i^d) = \top$ for some $0 \leq i \leq k$ and some $0 \leq d \leq d_j$. Again, we construct a run dag for A_q and $w^{(i)}$ starting with the node (q,i). As above, the constraints for δ always ensure the existence of a model for a node on some level of this run which creates the nodes on the following level. But note that the index d is increased in each transition from s_k to s_l. Since $\beta(|[s_q]|_L^{d_j+1}) = \bot$ is ensured by the constraints of the encoding, each infinite path in this run dag will eventually leave the component Q_j. By weakness, each infinite path proceeds into another component for which an accepting run dag has already been created by the induction hypothesis. Since a finite prefix of non-final states on any such an infinite path does not harm the acceptance condition, this run dag is accepting, too, and we have $w^{(i)}$ is accepted by A_q. □

Lemma 2. *Given a finite Kripke structure M and a WABA A, if there is an initialised infinite path π through M such that the corresponding word w is accepted by A then there is a $k \in \mathbb{N}$ such that $|[M, A_{WABA}]|_k$ is satisfiable.*

[9] According to this, implications from left to right instead of bi-implications in the constraints for δ would already suffice. It is also not hard to see that this does not destroy completeness: if there is an assignment satisfying the bi-implications then this assignment would also satisfy the weaker implications.

Proof. Suppose there is an infinite path π such that the corresponding word w is accepted by A. Since the class of languages accepted by weak alternating Büchi automata are the ω-regular languages we can without loss of generality assume π to be a (k,l)-loop for some $0 < l \leq k$. Furthermore, without loss of generality we can assume that π is minimal in the following sense. There is no infinite path π' through M such that the corresponding word w' is accepted by A and π' is a (k',l')-loop for some $k' < k$ and some l'.

It remains to be seen that $|[M, A_{WABA}]|_k$ is satisfiable. Hence, we need to construct a truth assignment β to the variables $s_0 s_1 \ldots s_k$, InLoop_i for each $0 \leq i \leq k$ as well as $|[s_q]|_i^d$ for each component Q_j of A, each $q \in Q_j$, each $0 \leq d \leq d_j$, and each $0 \leq i \leq k+1$. Note that the values of the other variables are determined by the values of these.

The values for the former are immediately given by the (k,l)-loop w. This shows satisfaction of the conjuncts $|[M]|_k$ and $|[\mathit{LoopConstraints}]|_k$.

For the rest of the variables we only give a proof sketch due to space considerations. After fixing w we can see A as a WABA *tree automaton* running on word (degenerate tree) w. Simplifying the encoding of δ with the values given by w to variables in the first phase above implements the tree WABA product construction in similar fashion as in Section 3.2 of [18] and thus the rest of the encoding solves the 1-letter WABA emptiness problem of a 1-letter product WABA induced by w. Now the rest of the encoding is basically a SAT implementation of a variant of the fixpoint computation algorithm of Theorem 4.7 in [18] to solve the 1-letter emptiness problem for WABAs. The non-accepting components correspond to least fixpoints and the accepting components correspond to greatest fixpoints. We can do an induction which processes one component at a time as in the proof of soundness above.

For an accepting component Q_j the values $|[s_q]|_i^0$ can be set to be identical to the final values computed by the algorithm of Theorem 4.7 in [18], thus obtaining a fixpoint which is easily checked to be a satisfying truth assignment.

For a non-accepting component Q_j the values $|[s_q]|_i^0$ can also be set to be identical to the final values computed by the algorithm of Theorem 4.7 in [18]. However, the values of $|[s_q]|_i^d$ with $1 \leq d \leq d_j$ are set to be the values obtained by a fixpoint approximation procedure which starts from the initial values given by $\beta(|[s_q]|_L^{d_j+1}) = \bot$ and for all i,d pairs proceeds for i from $k+1$ towards 0, and for d from d_j towards 1. It is easy to check that after at most $d_j = |Q_j|$ iterations through the loop in the backward direction final values have been obtained at the loop point $i = l, d = 1$ (recall that w is fixed and thus also the simplified form of δ is monotone and fixed according to w at each point of computing the fixpoint approximations), and thus we obtain a satisfying truth assignment for all the constraints concerning non-accepting components.

By the above and the fact that the algorithm of Theorem 4.7 in [18] computes $\top$ to the initial state iff w is accepted by A, we finally obtain $\beta(|[s_{q_0}]|_0^0) = \top$, and thus all constraints of the encoding are satisfied. $\square$

As a consequence of the proof, the encoding detects witnesses π that are (k,l)-loops at minimal parameter k.