Antichains: Alternative Algorithms for LTL Satisfiability and Model-Checking*

M. De Wulf[1], L. Doyen[2], N. Maquet[1]** and J.-F. Raskin[1]

[1] CS, Université Libre de Bruxelles (ULB), Belgium
[2] I&C, Ecole Polytechnique Fédérale de Lausanne (EPFL), Switzerland

Abstract. The linear temporal logic (LTL) was introduced by Pnueli as a logic to express properties over the computations of reactive systems. Since this seminal work, there have been a large number of papers that have studied deductive systems and algorithmic methods to reason about the correctness of reactive programs with regard to LTL properties. In this paper, we propose new efficient algorithms for LTL satisfiability and model-checking. Our algorithms do not construct nondeterministic automata from LTL formulas but work directly with alternating automata using efficient exploration techniques based on antichains.

1 Introduction

A model for an LTL formula over a set P of propositions is an infinite word w over the alphabet $\Sigma = 2^P$. An LTL formula ϕ defines a set of words $[\![\phi]\!] = \{w \in \Sigma^\omega \mid w \models \phi\}$. The *satisfiability problem* for LTL asks, given an LTL formula ϕ, if $[\![\phi]\!]$ is empty. The *model-checking problem* for LTL asks, given an omega-regular language $\mathcal{L}$ (e.g., the set of all computations of a reactive system) and a LTL formula ϕ, if $\mathcal{L} \subseteq [\![\phi]\!]$.

The link between LTL and omega-regular languages is at the heart of the *automata-theoretic approach* to LTL [24]. Given a LTL formula ϕ, we can construct a nondeterministic Büchi automaton (NBW) $\mathcal{A}_\phi$ whose language, noted $\mathsf{L_b}(\mathcal{A}_\phi)$, corresponds exactly to the models of ϕ, i.e., $\mathsf{L_b}(\mathcal{A}_\phi) = [\![\phi]\!]$. This reduces the satisfiability and model-checking problems to automata-theoretic questions.

This elegant framework has triggered a large body of works that have been implemented in explicit state model-checking tools such as SPIN [19] and in symbolic state model-checking tools such as SMV [15] and NUSMV [2].

The translation from LTL to NBW is central to the automata-theoretic approach to model-checking. When done explicitly, this translation is *worst-case exponential*. Explicit translation is required for explicit state model-checking, while in the symbolic approach to LTL model-checking [3] the NBW is symbolically encoded using boolean constraints. In [18], Rozier and Vardi have extensively compared symbolic and explicit approaches to satisfiability checking using a large number of tools. From their experiments, the symbolic approach scales better.

* This research was supported by the Belgian FNRS grant 2.4530.02 of the FRFC project "Centre Fédéré en Vérification" and by the project "MoVES", an Interuniversity Attraction Poles Project of the Belgian Federal Government.
** This author is supported by a FNRS-FRIA grant.

Efficient algorithms to reason on large LTL formulas are highly desirable. First, as writing formal requirements is a difficult task, verifying consistency is an issue for which efficient satisfiability checking would be highly valuable. Second, when model-checking a system and especially in the "debugging" phase, we may want to check properties that are true only under a set of assumptions, in which case specifications are of the form $\rho_1 \wedge \rho_2 \wedge \cdots \wedge \rho_n \rightarrow \phi$, and are usually very large. The reader will find such large formulas for example in [1] and in the experiments reported here.

In this paper, we present a new approach to LTL satisfiability and model-checking. Our approach avoids the explicit translation to NBW and does not resort to pure boolean reasoning as in the symbolic approach. Instead, we associate to every LTL formula an alternating Büchi automaton over a symbolic alphabet (sABW) that recognizes the models of the formula. The use of alternation instead of nondeterminism, and of symbolic alphabets allows for the construction of compact automata (the number of states and symbolic transitions are linear in the size of the LTL formula). While this construction is well-known and is an intermediate step in several translators from LTL to explicit NBW [23], we provide a new efficient way to analyze sABW. This new algorithm is an extension of [7], where we have shown how to efficiently decide the emptiness problem for (non-symbolic) ABW. The efficiency of our new algorithm relies on avoiding the explicit construction of a NBW and on the existence of pre-orders that can be exploited to efficiently compute fixpoint expressions directly over the transition relation of ABW.

Contributions The three main contributions of the paper are as follows. First, we adapt the algorithm of [7] for checking emptiness of symbolic ABW. The algorithm in [7] enumerates the alphabet Σ, which is impractical for LTL where the alphabet $\Sigma = 2^P$ is of exponential size. To cope with this, we introduce a way to combine BDD-based techniques with antichain algorithms, taking advantage of the strengths of BDDs for boolean reasoning. Second, we extend the combination of BDDs and antichains to model-checking of LTL specifications over symbolic Kripke structures. In [7], only explicit-state models and specifications given as NBWs were handled. Third, we have implemented and extensively tested our new algorithms. While the previous evaluations of antichain algorithms [6, 7] were performed on randomly generated models, we experiment here our new algorithms on concrete (i.e., with a meaningful semantics as opposed to randomly generated instances) satisfiability and model-checking examples. Most of our examples are taken in [18] and [20] where they are presented as benchmarks to compare model-checking algorithms. Our new algorithms outperform standard classical symbolic algorithms of the highly optimized industrial-level tools like NuSMV for both satisfiability and model-checking.

Related works We review the recent related works about LTL satisfiability and model-checking. For many years, great efforts have been devoted to reduce the cost of the explicit translation from LTL to NBW (see e.g., [8, 9, 22, 4]). The existing translators are now very sophisticated and it is questionable that they can still be drastically improved. According to [18], the current explicit tools are suitable for relatively small formulas but do not scale well. Rozier and Vardi advocate the use of symbolic methods as defined in [3] and tools like NuSMV for LTL satisfiability checking. They can handle much larger formulas than explicit tools. Therefore, we compare our new algorithms with NuSMV on benchmarks proposed by Rozier and Vardi, with very good results.

In [20], Vardi *et al.* propose a *hybrid approach* to model-checking: the system is represented symbolically using BDDs and the LTL formula is translated explicitly as a NBW. Their method has the nice property to partition the usually huge symbolic state space into pieces associated to each state of the NBW (this heuristic is called *property-driven partitioning* in [18]). Our approach also gains from this interesting feature, but in contrast to Vardi *et al.*, we do not need the expensive construction of the explicit NBW from the LTL formula.

Structure of the paper The paper is structured as follows. In Section 2, we recall the definitions of LTL and ABW. In Section 3, we present a forward semi-symbolic algorithm for satisfiability checking of LTL and we evaluate its performance in Section 4. In Section 5, we present a similar algorithm for model-checking and we show in Section 6 that it has performances that are better than the best existing tools. We draw some conclusion in Section 7.

2 LTL and Alternating Automata

Linear Temporal Logic Given a finite set P of propositions, a *Kripke structure* over P is a tuple $\mathcal{K} = \langle Q, q_\iota, \rightarrow_\mathcal{K}, \mathcal{L} \rangle$ where Q is a finite set of states, $q_\iota \in Q$ is the initial state, $\rightarrow_\mathcal{K} \subseteq Q \times Q$ is a transition relation, and $\mathcal{L} : Q \rightarrow 2^P$ is a labeling function. A *run* of $\mathcal{K}$ is an infinite sequence $\rho = q_0 q_1 \dots$ such that $q_0 = q_\iota$ and for all $i \geq 0$, $(q_i, q_{i+1}) \in \rightarrow_\mathcal{K}$. Let $\mathcal{L}(\rho) = \mathcal{L}(q_0)\mathcal{L}(q_1)\dots$ and define the *language* of $\mathcal{K}$ as $\mathsf{L}(\mathcal{K}) = \{\mathcal{L}(\rho) \mid \rho \text{ is a run of } \mathcal{K}\}$.

The *LTL formulas* over P are defined by $\phi ::= p \mid \neg\phi \mid \phi \vee \phi \mid \mathcal{X}\phi \mid \phi\mathcal{U}\phi$ where $p \in P$. Given an infinite word $w = \sigma_0\sigma_1 \dots \in \Sigma^\omega$ where $\Sigma = 2^P$, and an LTL formula ϕ over P, we say that w *satisfies* ϕ (written $w \models \phi$) if and only if (recursively):

- $\phi \equiv p$ and $p \in \sigma_0$,
- or $\phi \equiv \neg\phi_1$ and $w \not\models \phi_1$,
- or $\phi \equiv \phi_1 \vee \phi_2$ and $w \models \phi_1$ or $w \models \phi_2$,
- or $\phi \equiv \mathcal{X}\phi_1$ and $\sigma_1\sigma_2 \dots \models \phi_1$,
- or $\phi \equiv \phi_1\mathcal{U}\phi_2$ and for some $k \in \mathbb{N}$, $\sigma_k\sigma_{k+1} \dots \models \phi_2$ and for all i, $0 \leq i < k$, $\sigma_i\sigma_{i+1} \dots \models \phi_1$.

Additional formulas such as true, false and $\phi_1 \wedge \phi_2$ can be derived from the definition in the usual way, as well as the following temporal operators: let $\Diamond\phi = \mathsf{true}\,\mathcal{U}\phi$, $\Box\phi = \neg\Diamond\neg\phi$ and $\phi_1\mathcal{R}\,\phi_2 = \neg(\neg\phi_1\mathcal{U}\,\neg\phi_2)$.

Let $[\![\phi]\!] = \{w \in \Sigma^\omega \mid w \models \phi\}$ be the *language* defined by the LTL formula ϕ. The *satisfiability-checking problem* asks, given an LTL formula ϕ whether $[\![\phi]\!] \neq \varnothing$ (if so, we say that ϕ is *satisfiable*). Given an LTL formula ϕ and a Kripke structure $\mathcal{K}$ over P, we say that $\mathcal{K}$ satisfies ϕ (written $\mathcal{K} \models \phi$) if and only if $\mathsf{L}(\mathcal{K}) \subseteq [\![\phi]\!]$, that is for all runs ρ of $\mathcal{K}$, we have $\mathcal{L}(\rho) \models \phi$. The *model-checking problem* asks, given a Kripke structure $\mathcal{K}$ and an LTL formula ϕ, whether $\mathcal{K} \models \phi$. Satisfiability and model-checking of LTL are both PSPACE-COMPLETE. The time complexity of LTL model-checking is linear in the number of states of the Kripke structure and exponential in the size of the LTL formula.

Symbolic Alternating Büchi Automata The automata-based approach to satisfiability is to transform the formula ϕ to an automaton $\mathcal{A}_\phi$ that defines the same language, and then to check the emptiness of $\mathsf{L_b}(\mathcal{A}_\phi)$. Similarly for model-checking, we check the emptiness of $\mathsf{L}(\mathcal{K}) \cap \mathsf{L_b}(\mathcal{A}_{\neg\phi})$. These automata are defined over a symbolic alphabet of propositional formulas. Intuitively, a transition labeled by a formula φ encodes all the transitions that are labeled by a set of propositions that satisfies φ.

Given a finite set Q, let $\mathsf{Lit}(Q) = Q \cup \{\neg q \mid q \in Q\}$ be the set of *literals* over Q, and $\mathcal{B}^+(Q)$ be the set of *positive boolean formulas* over Q, that is formulas built from elements in $Q \cup \{\mathsf{true}, \mathsf{false}\}$ using the boolean connectives $\wedge$ and $\vee$. Given $R \subseteq Q$ and $\varphi \in \mathcal{B}^+(Q)$, we write $R \models \varphi$ if and only if the truth assignment that assigns true to the elements of R and false to the elements of $Q \setminus R$ satisfies φ.

A *symbolic alternating Büchi automaton* (sABW) over the set of propositions P is a tuple $\mathcal{A} = \langle \mathsf{Loc}, I, \Sigma, \delta, \alpha \rangle$ where:

- Loc is a finite set of states (or locations);
- $I \in \mathcal{B}^+(\mathsf{Loc})$ defines the set of possible initial sets of locations. Intuitively, a set $s \subseteq \mathsf{Loc}$ is initial if $s \models I$;
- $\Sigma = 2^P$ is the alphabet;
- $\delta : \mathsf{Loc} \to \mathcal{B}^+(\mathsf{Lit}(P) \cup \mathsf{Loc})$ is the transition function. The use of formulas to label transitions in δ allows a compact representation of δ', e.g., using BDD. We write $\ell \xrightarrow{\sigma}_\delta s$ whenever $\sigma \cup s \models \delta(\ell)$;
- $\alpha \subseteq \mathsf{Loc}$ is the set of accepting states.

A *run* of $\mathcal{A}$ on an infinite word $w = \sigma_0\sigma_1 \cdots \in \Sigma^\omega$ is a DAG $T_w = \langle V, V_\iota, \to \rangle$ where:

- $V \subseteq \mathsf{Loc} \times \mathbb{N}$ is the set of nodes. A node (ℓ, i) represents the location ℓ after the first i letters of w have been read by A. Nodes of the form (ℓ, i) with $\ell \in \alpha$ are called *α-nodes*;
- $V_\iota \subseteq \mathsf{Loc} \times \{0\}$ is such that $V_\iota \subseteq V$ and $\{\ell \mid (\ell, 0) \in V_\iota\} \models I$;
- and $\to \subseteq V \times V$ is such that (i) if $(\ell, i) \to (\ell', i')$ then $i' = i + 1$ and (ii) $\sigma_i \cup \{\ell' \mid (\ell, i) \to (\ell', i+1)\} \models \delta(\ell)$ for all $(\ell, i) \in V$.

A run $T_w = \langle V, v_\iota, \to \rangle$ of $\mathcal{A}$ on an infinite word w is *accepting* if all its infinite paths visit α-nodes infinitely often. An infinite word $w \in \Sigma^\omega$ is *accepted* by $\mathcal{A}$ if there exists an accepting run on it. We denote by $\mathsf{L_b}(\mathcal{A})$ the set of infinite words accepted by $\mathcal{A}$.

A *nondeterministic Büchi automaton* (sNBW) is an sABW $\mathcal{A} = \langle \mathsf{Loc}, I, \Sigma, \delta, \alpha \rangle$ such that I is a disjunction of locations, and for each $\ell \in \mathsf{Loc}$, $\delta(\ell)$ is a disjunction of formulas of the form $\varphi \wedge \ell'$ where $\varphi \in \mathcal{B}^+(\mathsf{Lit}(P))$ and $\ell' \in \mathsf{Loc}$. In the sequel, we often identify I with the set of locations that appear in I and δ as a the set of all transitions (ℓ, σ, ℓ') such that $\sigma \cup \{\ell'\} \models \delta(\ell)$. Runs of sNBW reduce to (linear) sequences of locations as a single initial state can be chosen in I, and each node has at most one successor. We define the *reverse* automaton of $\mathcal{A}$ as the sNBW $\mathcal{A}^{-1} = \langle \mathsf{Loc}, \alpha, \Sigma, \delta^{-1}, I \rangle$ where $\delta^{-1} = \{(\ell, \sigma, \ell') \mid (\ell', \sigma, \ell) \in \delta\}$.

There exists a simple translation from LTL to sABW [9, 23]. We do not recall the translation but we give an example hereafter. The construction is defined recursively over the structure of the formula and it gives a compact automata representation of LTL formulas (the number of states of the automaton is linear in the size of the formula). The succinctness results from the presence of alternation in the transition function, and from the use of propositional formulas (the symbolic alphabet) to label the transitions.

Example Fig. 1 shows the sABW for the negation of the formula $\Box\Diamond p \rightarrow \Box(\neg p \rightarrow \Diamond r)$, which is equivalent to the conjunction of $\phi_1 \equiv \Box\Diamond p$ and $\phi_2 \equiv \Diamond(\neg p \wedge \Box\neg r)$. The accepting states are $\alpha = \{\ell_1, \ell_4\}$. Intuitively, the states ℓ_4, ℓ_3 check that ϕ_1 holds, and states ℓ_2, ℓ_1 check that ϕ_2 holds. The conjunction is enforced by the initial condition $\ell_4 \wedge \ell_2$. We write transitions in disjunctive normal form and we consider two parts in each conjunction, one for the propositions and one for the locations. E.g., the transition from ℓ_4 is $\delta(\ell_4) = (p \wedge \ell_4) \vee (\mathsf{true} \wedge \ell_3 \wedge \ell_4)$, and from ℓ_3 it is $\delta(\ell_3) = (\mathsf{true} \wedge \ell_3) \vee (p \wedge \mathsf{true})$. We use true to emphasize when there is no constraint on either propositions or locations. In the figure, a conjunction of locations is depicted by a forked arrow, the conjunction of literals is labelling the arrow. One arrow from ℓ_3 has an empty target as $\varnothing \models \mathsf{true}$. If the control of the automaton is in ℓ_4 and the automaton reads some $\sigma \in 2^P$ such that $p \notin \sigma$, then the control moves simultaneously to location ℓ_4 and location ℓ_3. As ℓ_3 is not accepting, the control has to leave ℓ_3 eventually by reading some σ' such that $p \in \sigma'$. So every run accepted from ℓ_4 satisfies $\Box\Diamond p$.

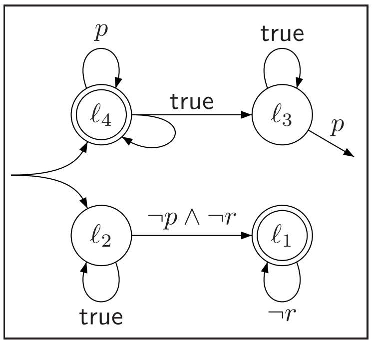


Fig. 1. Alternating automaton for $\varphi \equiv \neg(\Box\Diamond p \rightarrow \Box(\neg p \rightarrow \Diamond r))$.

3 Satisfiability-Checking of LTL

By the above translation from LTL to sABW, the satisfiability-checking problem reduces to emptiness of sABW (that is to decide, given an sABW $\mathcal{A}$, whether $\mathsf{L_b}(\mathcal{A}) = \varnothing$) which can be solved using a translation from sABW to sNBW that preserves the language of the automaton [16]. This construction involves an exponential blow-up that makes straight implementations infeasible in practice. We do not construct this automaton, but the correctness of our approach relies on its existence.

Miyano-Hayashi construction The construction transforms an sABW into a sNBW that accepts the same language. It has the flavor of the subset construction for automata over finite words. Intuitively, the sNBW maintains a set s of states of the sABW that corresponds to a whole level of a guessed run DAG of the sABW. In addition, the sNBW maintains a set o of states that "owe" a visit to an accepting state. Whenever the set o gets empty, meaning that every path of the guessed run has visited at least one accepting state, the set o is initiated with the current level of the guessed run. The Büchi condition asks that o gets empty infinitely often in order to ensure that every path of the run DAG visits accepting states infinitely often. The construction is as follows (we adapt it for symbolic sABW).

Given an sABW $\mathcal{A} = \langle \mathsf{Loc}, I, \Sigma, \delta, \alpha \rangle$ over P, let $\mathsf{MH}(\mathcal{A}) = \langle Q, I^{\mathsf{MH}}, \Sigma, \delta^{\mathsf{MH}}, \alpha^{\mathsf{MH}} \rangle$ be a sNBW where:

- $Q = 2^{\mathsf{Loc}} \times 2^{\mathsf{Loc}}$;
- I^{MH} is the disjunction of all the pairs $\langle s, \varnothing \rangle$ such that $s \models I$;
- δ^{MH} is defined for all $\langle s, o \rangle \in Q$ as follows:
  - If $o \neq \varnothing$, then $\delta^{\mathsf{MH}}(\langle s, o \rangle)$ is the disjunction of all the formulas $\varphi \wedge \langle s', o' \setminus \alpha \rangle$ with $\varphi \in \mathcal{B}^+(\mathsf{Lit}(P))$ such that:
    (i) $o' \subseteq s'$;
    (ii) $\forall \ell \in s \cdot \forall \sigma \subseteq P :$ if $\sigma \models \varphi$ then $\sigma \cup s' \models \delta(\ell)$;
    (iii) $\forall \ell \in o \cdot \forall \sigma \subseteq P :$ if $\sigma \models \varphi$ then $\sigma \cup o' \models \delta(\ell)$.
  - If $o = \varnothing$, then $\delta^{\mathsf{MH}}(\langle s, o \rangle)$ is the disjunction of all the formulas $\varphi \wedge \langle s', s' \setminus \alpha \rangle)$ with $\varphi \in \mathcal{B}^+(\mathsf{Lit}(P))$ such that:
    $\forall \ell \in s \cdot \forall \sigma \subseteq P :$ if $\sigma \models \varphi$ then $\sigma \cup s' \models \delta(\ell)$;
- $\alpha^{\mathsf{MH}} = 2^{\mathsf{Loc}} \times \{\varnothing\}$.

The number of states of the Miyano-Hayashi construction is exponential in the number of states of the original automaton.

Theorem 1 ([16]) *For all sABW $\mathcal{A}$, we have* $\mathsf{L_b}(\mathsf{MH}(\mathcal{A})) = \mathsf{L_b}(\mathcal{A})$.

Fixpoint formulas To check the satisfiability of an LTL formula ϕ we check the emptiness of $\mathsf{MH}(\mathcal{A}_\phi) = \langle Q, I^{\mathsf{MH}}, \Sigma, \delta^{\mathsf{MH}}, \alpha^{\mathsf{MH}} \rangle$.

It is well-known that $[\![\phi]\!]= \mathsf{L_b}(\mathcal{A}_\phi) = \varnothing$ iff $I^{\mathsf{MH}} \cap \mathcal{F}_\phi = \varnothing$ where $\mathcal{F}_\phi$ is the following fixpoint formula [5]:

$$\mathcal{F}_\phi \equiv \nu y \cdot \mu x \cdot (\mathsf{Pre}(x) \cup (\mathsf{Pre}(y) \cap \alpha^{\mathsf{MH}}))$$

where $\mathsf{Pre}(L) = \{q \in Q \mid \exists \sigma \in \Sigma \cdot \exists q' \in L : \sigma \cup \{q'\} \models \delta^{\mathsf{MH}}(q)\}$.

We call $\mathcal{F}_\phi$ a *backward* algorithm as it uses the predecessor operator $\mathsf{Pre}(\cdot)$. The set of states that are computed in the iterations of the fixpoints may be unreachable from the initial states [12]. Therefore, a *forward* algorithm based on the successor operator $\mathsf{Post}(\cdot)$ would have the advantage of exploring only the reachable states of the automaton. Moreover, the number of successors is often smaller than the number of predecessors, especially when the LTL formula "specifies" initial conditions that reduce the forward non-determinism.

The following fixpoint formulas compute the accepting reachable states R_α and then the set $\mathcal{F}'_\phi$ in a forward fashion.

$$R_\alpha \equiv \alpha^{\mathsf{MH}} \cap \mu x \cdot (\mathsf{Post}(x) \cup I^{\mathsf{MH}})$$
$$\mathcal{F}'_\phi \equiv \nu y \cdot \mu x \cdot (\mathsf{Post}(x) \cup (\mathsf{Post}(y) \cap R_\alpha))$$

where $\mathsf{Post}(L) = \{q \in Q \mid \exists \sigma \in \Sigma \cdot \exists q' \in L : \sigma \cup \{q\} \models \delta^{\mathsf{MH}}(q')\}$.

Theorem 2 $\mathsf{L_b}(\mathcal{A}_\phi) = \varnothing$ *iff* $\mathcal{F}'_\phi = \varnothing$.

Proof. Define $\Delta^{\mathsf{MH}} : Q \times \Sigma^+$ the extension of the transition relation δ^{MH} to nonempty words as follows (recursively): $\Delta^{\mathsf{MH}}(q,\sigma) = \delta^{\mathsf{MH}}(q,\sigma)$ and $\Delta^{\mathsf{MH}}(q,w\sigma) = \{q' \in Q \mid \exists q'' \in \Delta^{\mathsf{MH}}(q,w) : \sigma \cup \{q'\} \models \delta^{\mathsf{MH}}(q'')\}$ for each $q \in Q, w \in \Sigma^+$ and $\sigma \in \Sigma$.

Let $\mathcal{C}^{\mathsf{MH}} = \{q \in Q \mid \exists w \in \Sigma^+ : q \in \Delta^{\mathsf{MH}}(q,w)\}$ be the set of *looping* states in $\mathsf{MH}(\mathcal{A}_\phi)$. From the definition of Büchi acceptance condition for NBW, we have $\mathsf{L_b}(\mathcal{A}_\phi) = \varnothing$ iff $\mathcal{C}^{\mathsf{MH}} \cap R_\alpha = \varnothing$. Let $\mathsf{HM}(\mathcal{A}_\phi)$ be the reverse automaton[3]. The following equivalences (the first one being well-known) establish the theorem:

$$\begin{array}{ll} & \nu y \cdot \mu x \cdot (\mathsf{Pre}(x) \cup (\mathsf{Pre}(y) \cap R_\alpha)) = \varnothing \\ \text{iff} & \mathcal{C}^{\mathsf{MH}} \cap R_\alpha = \varnothing \\ \text{iff} & \mathcal{C}^{\mathsf{HM}} \cap R_\alpha = \varnothing \\ \text{iff} & \nu y \cdot \mu x \cdot (\mathsf{Post}(x) \cup (\mathsf{Post}(y) \cap R_\alpha)) = \varnothing \\ \text{iff} & \mathcal{F}'_\phi = \varnothing \end{array}$$

■

Closed Sets and Antichains Remember that Q is exponential in the size of ϕ. Following the lines of [7], we show that $\mathcal{F}'_\phi$ can be computed more efficiently. Let $\preceq \subseteq Q \times Q$ be a preorder and let $q_1 \prec q_2$ iff $q_1 \preceq q_2$ and $q_2 \not\preceq q_1$. A set $R \subseteq Q$ is $\preceq$*-closed* iff for all $q_1, q_2 \in Q$, if $q_1 \preceq q_2$ and $q_2 \in R$ then $q_1 \in R$. The $\preceq$*-closure* of R, is the set $[\![R]\!]_\preceq = \{q \in Q \mid \exists q' \in R : q \preceq q'\}$. Let $\lceil R \rceil_\preceq = \{q \in R \mid \nexists q' \in R : q \prec q'\}$ be the set of $\preceq$*-maximal elements* of R, and dually let $\lfloor R \rfloor_\succeq = \{q \in R \mid \nexists q' \in R : q \succ q'\}$ be the set of $\succeq$*-minimal elements* of R.

For all $\preceq$-closed sets $R \subseteq Q$, we have $R = [\![\lceil R \rceil_\preceq]\!]_\preceq$ and for all $\succeq$-closed sets $R \subseteq Q$, we have $R = [\![\lfloor R \rfloor_\preceq]\!]_\succeq$. Furthermore, if $\preceq$ is a partial order, then $\lceil R \rceil_\preceq$ is an *antichain* and it is a canonical representation of R.

Let $\mathcal{A} = \langle \mathsf{Loc}, I, \Sigma, \delta, \alpha \rangle$ be a NBW. A preorder $\preceq \subseteq \mathsf{Loc} \times \mathsf{Loc}$ is a *forward-simulation* for $\mathcal{A}$ (q_1 forward-simulates q_2 if $q_1 \preceq q_2$) if for all $q_1, q_2, q_3 \in \mathsf{Loc}$, for all $\sigma \in \Sigma$, (i) if $q_1 \preceq q_2$ and $q_2 \xrightarrow{\sigma}_\delta q_3$ then there exists $q_4 \in \mathsf{Loc}$ such that $q_1 \xrightarrow{\sigma}_\delta q_4$ and $q_4 \preceq q_3$, and (ii) if $q_1 \preceq q_2$ and $q_2 \in \alpha$ then $q_1 \in \alpha$. A *backward-simulation* for $\mathcal{A}$ is a forward-simulation for $\mathcal{A}^{-1}$. It is not true in general that $\succeq$ is a backward-simulation for $\mathcal{A}$ if $\preceq$ is a forward-simulation for $\mathcal{A}$ (consider a state q_c that has no predecessor and such that $q_b \preceq q_c$). However, the following lemma shows that the language of a sNBW is unchanged if we add a transition from a state q_a to a state q_c which is forward-simulated by one of the successors of q_a. By adding in this way all the possible transitions, we obtain a sNBW for which $\succeq$ is a backward-simulation.

Lemma 3 *Let $\mathcal{A}$ be a sNBW with transition relation $\delta_\mathcal{A}$ and $\preceq$ be a forward-simulation relation for $\mathcal{A}$. If $(q_a, \sigma, q_b) \in \delta_\mathcal{A}$ and $q_b \preceq q_c$, then the sNBW $\mathcal{A}'$ that differs from $\mathcal{A}$ only by its transition relation $\delta_{\mathcal{A}'} = \delta_\mathcal{A} \cup \{(q_a, \sigma, q_c)\}$ defines the same language as $\mathcal{A}$, that is $\mathsf{L_b}(\mathcal{A}') = \mathsf{L_b}(\mathcal{A})$.*

As a dual of the results of [7], it is easy to show that given a backward-simulation $\succeq$ for $\mathsf{MH}(\mathcal{A}_\phi)$, all the sets that are computed to evaluate R_α and $\mathcal{F}'_\phi$ are $\succeq$-closed, that is

[3] In the sequel, $\mathsf{Pre}(\cdot)$ and $\mathsf{Post}(\cdot)$ are always computed on $\mathsf{MH}(\mathcal{A}_\phi)$ and never on $\mathsf{HM}(\mathcal{A}_\phi)$.

I^{MH} and α^{MH} are $\succeq$-closed, and $x \cap y$, $x \cup y$ and $\mathsf{Pre}(x)$ are $\succeq$-closed whenever x and y are $\succeq$-closed [7]. The relation $\preceq_{\mathsf{alt}}$ defined by $\langle s, o\rangle \preceq_{\mathsf{alt}} \langle s', o'\rangle$ iff (i) $s \subseteq s'$, (ii) $o \subseteq o'$, and (iii) $o = \varnothing$ iff $o' = \varnothing$ is a forward-simulation for $\mathsf{MH}(\mathcal{A}_\phi)$. Therefore, the relation $\succeq_{\mathsf{alt}}$ (which is $\preceq_{\mathsf{alt}}^{-1}$) is a backward-simulation if we modify the transition relation of $\mathsf{MH}(\mathcal{A}_\phi)$ as follows: if $\delta^{\mathsf{MH}}(\langle s, o\rangle)$ is a disjunction of formulas of the form $\varphi \wedge \langle s', o'\rangle$ with $\varphi \in \mathcal{B}^+(\mathsf{Lit}(P))$, then we disjunctively add all the formulas $\varphi \wedge \langle s'', o''\rangle$ to $\delta^{\mathsf{MH}}(\langle s, o\rangle)$ such that $\langle s', o'\rangle \preceq_{\mathsf{alt}} \langle s'', o''\rangle$. According to Lemma 3, this preserves the language of $\mathsf{MH}(\mathcal{A}_\phi)$. We keep only the $\succeq_{\mathsf{alt}}$-minimal elements of $\succeq_{\mathsf{alt}}$-closed sets to evaluate $\mathcal{F}'_\phi$ and so we dramatically reduce the size of the sets that are handled by the algorithms.

Remark 1. The intuition for keeping only minimal elements is as follows. Let A be a sABW, along a run of $\mathsf{MH}(A)$ that reads a word w, a pair $\langle s, o\rangle$ keeps track of the set of locations from which the sABW has to accept the suffix and to pass by accepting states. Clearly, if there is no accepting run from $\langle s, o\rangle$ then there is no accepting run from any pair $\langle s', o'\rangle$ where $\langle s', o'\rangle \succeq_{\mathsf{alt}} \langle s, o\rangle$. In short, the antichain algorithm concentrates on the most promising pairs that can be part of an accepting run by only keeping track of minimal elements.

Elements of efficient implementation The efficient computation of the $\succeq_{\mathsf{alt}}$-minimal elements of $\mathsf{Post}(\llbracket\cdot\rrbracket_{\succeq_{\mathsf{alt}}})$ is not trivial. For instance, the algorithm of [7] would have to enumerate all the truth assignments of propositional formulas over P appearing on transitions. To mitigate this problem, we propose to combine BDDs and antichains as follows. Antichains of pairs $\langle s, o\rangle$ are represented *explicitly* (as a list of pairs of sets of locations) while computation of the successors of a pair $\langle s, o\rangle$ is done *symbolically*. This is why, in the following, we call our algorithm *semi-symbolic*.

Given a BDD B over a set of variables V (seen as a boolean formula over V), let $\llbracket B\rrbracket$ be the set of truth assignments over V that satisfy B. Given a pair $\langle s, o\rangle$, Algorithm 1 computes the set $L_{\mathsf{Post}} = \lfloor \mathsf{Post}(\llbracket\{\langle s, o\rangle\}\rrbracket_{\succeq_{\mathsf{alt}}})\rfloor_{\succeq_{\mathsf{alt}}}$. When computing the successors of a pair $\langle s, o\rangle$, the algorithm uses the intermediate boolean variables $x_1, \dots, x_n$, $y_1, \dots, y_n$ and $y'_1, \dots, y'_n$ to encode respectively the sets s', $o' \setminus \alpha$ and o' where $\langle s', o' \setminus \alpha\rangle \in \mathsf{Post}(\llbracket\{\langle s, o\rangle\}\rrbracket_{\succeq_{\mathsf{alt}}})$. We write $\delta(\ell)[x_1 \dots x_n]$ to denote the formula $\delta(\ell)$ in which each occurrence of a location ℓ_i is replaced by variable x_i for all $1 \leq i \leq n$. The computations at lines 1–6 match exactly the definition of the Miyano-Hayashi construction. The BDD $\theta(y, y')$ is used to remove the accepting states from the sets o' in B_L, and the existential quantification over the set P of propositions matches the definition of the $\mathsf{Post}(\cdot)$ operator. Then, using a BDD $\omega(x, y, x', y')$ that encodes the relation $\prec_{\mathsf{alt}}$ (we have $\langle s', o'\rangle \prec_{\mathsf{alt}} \langle s, o\rangle$ in ω where s, o, s', o' are encoded respectively with variables x, y, x', y'), we eliminate the non-minimal elements in B_L and we reconstruct the explicit set of pairs $\langle s', o'\rangle$ from $B_L^{\min}(x, y)$.

The encoding that we have chosen uses a number of variables linear in the size of the set of locations of the sABW and number of propositions. Preliminary experiments have shown that this approach is faster than an enumerative algorithm implemented in a clever way. The combinatorial blow-up that is hidden in the quantification $\exists P$ over propositions is likely to be the reason for this, as it is well known that for this purpose symbolic algorithms are faster in practice.

Algorithm 1: Semi-symbolic Algorithm for $\mathsf{Post}(\cdot)$.

Data : An sABW $\mathcal{A} = \langle \mathsf{Loc}, I, \Sigma, \delta, \alpha \rangle$, and a pair $\langle s, o \rangle$ such that $o \subseteq s$.
Result : The set $L_{\mathsf{Post}} = \lfloor \mathsf{Post}([\![\{\langle s, o \rangle\}]\!]_{\succeq_{\mathsf{alt}}}) \rfloor_{\succeq_{\mathsf{alt}}}$.
begin

if $o \neq \varnothing$ **then**

$$B_L(x, y') \leftarrow \exists P : \begin{cases} \bigwedge_{i=1}^{n} y'_i \to x_i & // \ \mathsf{o}' \subseteq \mathsf{s}' \\ \wedge \bigwedge_{\ell \in s} \delta(\ell)[x_1 \ldots x_n] & // \ \sigma \cup \mathsf{s}' \models \delta(\ell) \text{ for all } \ell \in \mathsf{s} \\ \wedge \bigwedge_{\ell \in o} \delta(\ell)[y'_1 \ldots y'_n] & // \ \sigma \cup \mathsf{o}' \models \delta(\ell) \text{ for all } \ell \in \mathsf{o} \end{cases}$$

$$\theta(y, y') \leftarrow \bigwedge_{\ell_i \in \alpha} \neg y_i \wedge \bigwedge_{\ell_i \notin \alpha} y_i \leftrightarrow y'_i \qquad // \ \mathsf{o}' \setminus \alpha$$

$$B_L(x, y) \leftarrow \exists\, y' : B_L(x, y') \wedge \theta(y, y')$$

else

$$B_L(x, y) \leftarrow \exists\, P : \begin{cases} \bigwedge_{\ell_i \in \alpha} \neg y_i \wedge \bigwedge_{\ell_i \notin \alpha} y_i \leftrightarrow x_i & // \ \mathsf{o}' \text{ is } \mathsf{s}' \setminus \alpha \\ \wedge \bigwedge_{\ell \in s} \delta(\ell)[x_1 \ldots x_n] & // \ \mathsf{s}' \models \delta(\ell) \text{ for all } \ell \in \mathsf{s} \end{cases}$$

$$\omega(x, y, x', y') \leftarrow \begin{cases} \bigwedge_{i=1}^{n} \big(x'_i \to x_i \wedge y'_i \to y_i\big) \\ \wedge \bigvee_{i=1}^{n} \big(x_i \neq x'_i \vee y_i \neq y'_i\big) & // \ \omega \text{ encodes } \prec_{\mathsf{alt}} \\ \wedge \big(\bigvee_{i=1}^{n} y_i\big) \leftrightarrow \big(\bigvee_{i=1}^{n} y'_i\big) \end{cases}$$

$$B_L^{\min}(x, y) \leftarrow B_L(x, y) \wedge \neg\big(\exists x', y' : \omega(x, y, x', y') \wedge B_L(x', y')\big)$$

$$L_{\mathsf{Post}} \leftarrow \big\{\langle s', o' \rangle \mid \exists v \in [\![B_L^{\min}]\!] \colon s' = \{\ell_i \mid v(x_i) = \mathsf{true}\}, o' = \{\ell_i \mid v(y_i) = \mathsf{true}\}\big\}$$

end

4 Satisfiability: Performance Evaluation

We have implemented our new forward semi-symbolic satisfiability algorithm in a prototype written in Python[4]. Before evaluating the fixpoint expression, the prototype performs the following steps: the LTL formula is parsed, standard fast heuristical rewriting rules are applied [22], and the formula is then translated to a sABW [9]. This sABW contains n locations, where n is linear in the size of the LTL formula. To compactly represent the symbolic transitions associated to each location, we use BDDs over $n + k$ boolean variables where k is the number of propositions that appear in the formula. Usually, the BDDs that are associated to the locations of the sABW are small because they are typically expressing constraints over few locations. This is usually in sharp contrast with the size of the BDDs that represent the underlying NBW of a LTL formula in fully-symbolic model-checking. The BDD package used by our prototype is CUDD [21] which is available through a python binding called PYCUDD[5].

Comparison with the state-of-the-art algorithms According to the extensive survey of Vardi and Rozier [18] NUSMV is the most efficient available tool for LTL satisfiability. We therefore compare our prototype with NUSMV. Satisfiability checking with NuSMV is done simply by model checking the negation of the formula against a uni-

[4] Python is an interpreted object-oriented language. See http://www.python.org
[5] http://www.ece.ucsb.edu/bears/pycudd.html

versal Kripke structure. In all our experiments, we used NuSMV 2.4 with the default options. [6] No variable reordering techniques were activated in either tool.

Benchmarks. We have compared both tools on four families of LTL formulas. Our satisfiability-checking prototype is reported as "sat.py" in the figures. All the experiments were performed on a single Intel Xeon CPU at 3.0 GHz, with 4 GB of RAM, using a timeout of 10 min and a maximum memory usage limit of 2.5 GB (all experiments timed out before exceeding the memory limit). All the LTL formulas tested here can be found in the long version of this paper[14].[7]

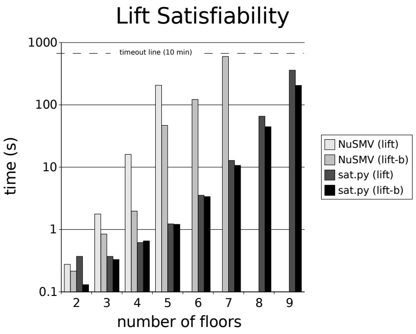
The first family is a parametric specification of a lift system with n floors that we have taken from Harding's thesis [11]. Two encodings are used: one ("lift") that uses a linear number of variables per floor, and another ("lift-b") which uses a number of variables that is logarithmic in the number of floors (resulting in larger formulas). As seen in figure 2(a), our algorithm scales much better than NuSMV for both encodings. For more than 7-floor (a formula with 91 temporal operators and 17 distinct boolean propositions), NuSMV is more than 60 times slower than our tool.

The second family of formulas was referenced in [20] as examples of difficult LTL to NBW translation and describes liveness properties for the Szymanski mutual exclusion protocol and for a variant of this protocol due to Pnueli. We have run both our prototype and NuSMV on these four formulas (pos) and their negation (neg), all of which can be found in [14]. Again, our tool shows better performances (by factors of 50 and higher), as reported in figure 2(b).

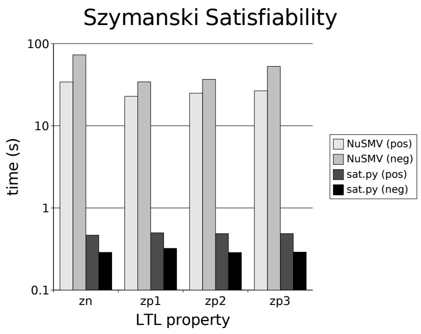
The third family we used is a random model described in [4] and also used in [18]. Random LTL formulas are generated according to the following parameters: the length of the formula (L), the number of propositions (N) each with equal probability of occurrence, and the probability (P) of choosing a temporal operator ($\mathcal{U}$ or $\mathcal{R}$). As in [18], we fix $P = 0.5$ and compare execution times for $L \in \{10, 20, \ldots, 100\}$ and for both $N = 2$ and $N = 4$. As indicated by figure 2(c), our algorithm copes much better with the joint increase in formula length and number of propositions[8]. For $L = 100$, going from $N = 2$ to $N = 4$ multiplies the time needed by NuSMV by 7, while our prototype only exhibits an 8% increase in execution time.

Finally, the last set of formulas (also taken in [18]) describes how a binary counter, parameterized by its length, is incremented. Two ways of formalizing the increment are considered ("count", "count-l"). Those formulas are quite particular as they all define a unique model: for $n = 2$, the model is $(00\cdot 01\cdot 10\cdot 11)^\omega$. In this benchmark, the classical fully-symbolic algorithm behaves much better than our antichain algorithm. This is not surprising for two reasons. First, the efficiency of our antichain-based algorithms comes from the ability to identify prefixes of runs in the ABW which can be ignored because they impose more constraints than others on the future (see Remark 1). As there is only one future allowed by the formula, the locations of the NBW defined by the Miyano-Hayashi construction are incomparable for the simulation relation defined in

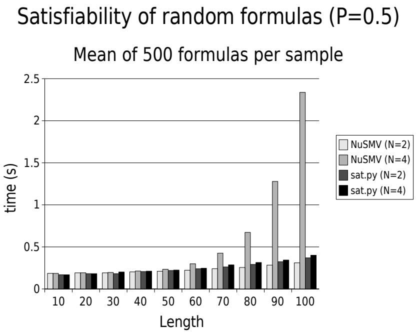
[6] The options are numerous, check the NuSMV user manual for full details.
[7] They can also be downloaded at http://www.ulb.ac.be/di/ssd/nmaquet/tacas.
[8] We report only the mean execution times, but the standard deviation is similar for both tools.



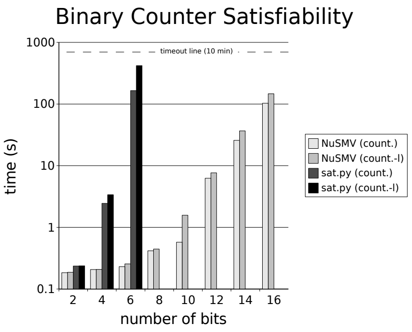
(a)



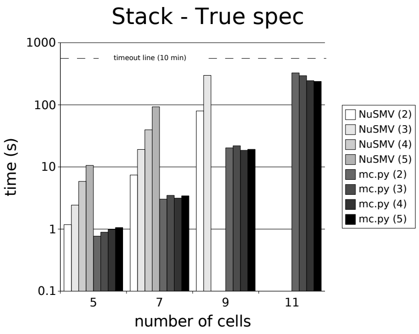
(b)



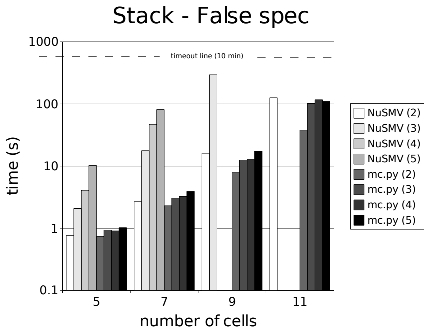
(c)



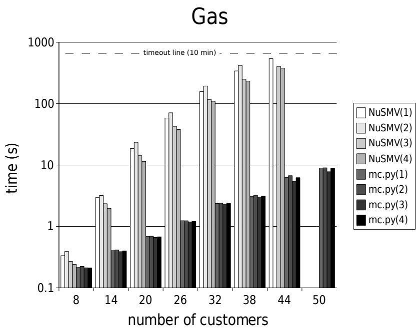
(d)



(e)



(f)



(g)

Time and Memory for the Bakery Algorithm				
	Mutual Exclusion		Fairness	
	NuSMV	mc.py	NuSMV	mc.py
2 proc.	0.22s 10.7MB	0.55s 60.3MB	10.66s 35.4MB	3.47s 60.3MB
3 proc.	359.12s 656.7MB	8.81s 87.3MB	28740.17s 495.0MB	730.60s 201.3MB
4 proc.	> 1000s out of Mem.	630, 7s 579, 01MB	N/A N/A	N/A N/A

(h)

Fig. 2. Experimental results comparing NuSMV with our algorithms.

Section 3, causing very long antichains and poor performances. This can be considered as a pathological and maybe not very interesting case.

5 LTL Model-Checking

Our algorithm for LTL model-checking is based on forward exploration and semi-symbolic representations. It is also related to the hybrid approach proposed by Vardi *et al.* in [20] with the essential difference that we work directly on the sABW for the LTL formula, avoiding the construction of a NBW.

Given a Kripke structure $\mathcal{K} = \langle Q, q_\iota, \rightarrow_\mathcal{K}, \mathcal{L}\rangle$ and an LTL formula ϕ, the model-checking problem for $\mathcal{K}$ and ϕ reduces to the emptiness of $\mathsf{L}(\mathcal{K}) \cap \mathsf{L}_\mathsf{b}(\mathcal{A}_{\neg\phi})$ (where $\mathcal{A}_{\neg\phi} = \langle \mathsf{Loc}, I, \Sigma, \delta, \alpha\rangle$ is the sABW for $\neg\phi$) which can be checked by computing the following fixpoint formulas over the lattice of subsets of $Q \times 2^{\mathsf{Loc}} \times 2^{\mathsf{Loc}}$:

$$R^{\mathcal{K}}_\alpha \equiv \alpha' \cap \mu x \cdot (\mathsf{Post}_{\mathsf{MC}}(x) \cup I')$$
$$\mathcal{F}^{\mathcal{K}}_\phi \equiv \nu y \cdot \mu x \cdot (\mathsf{Post}_{\mathsf{MC}}(x) \cup (\mathsf{Post}_{\mathsf{MC}}(y) \cap R^{\mathcal{K}}_\alpha))$$

where $\mathsf{Post}_{\mathsf{MC}}(L) = \{(q', \langle s', o'\rangle) \mid \exists (q, \langle s, o\rangle) \in L : q \rightarrow_\mathcal{K} q' \wedge \mathcal{L}(q) \cup \{\langle s', o'\rangle\} \models \delta^{\mathsf{MH}}(\langle s, o\rangle)\}$, $I' = \{q_\iota\} \times I^{\mathsf{MH}}$ and $\alpha' = Q \times \alpha^{\mathsf{MH}}$ (where the MH superscript refers to the sABW $\mathsf{MH}(\mathcal{A}_{\neg\phi})$). As before, we have $\mathcal{F}^{\mathcal{K}}_\phi = \varnothing$ iff $\mathsf{L}(\mathcal{K}) \cap \mathsf{L}_\mathsf{b}(\mathcal{A}_{\neg\phi}) = \varnothing$ iff $\mathcal{K} \models \phi$.

Moreover, there exists a partial order $\succeq_{\mathsf{MC}}$ for which all the sets that are computed to evaluate $\mathcal{F}^{\mathcal{K}}_\phi$ are $\succeq_{\mathsf{MC}}$-closed. The relation $\succeq_{\mathsf{MC}}$ is defined by $(q, \langle s, o\rangle) \succeq_{\mathsf{MC}} (q', \langle s', o'\rangle)$ iff $q = q'$ and $\langle s, o\rangle \succeq_{\mathsf{alt}} \langle s', o'\rangle$.

We use a semi-symbolic forward algorithm for model-checking as this is the most promising combination, in the light of our experiences with satisfiability. We assume a symbolic representation of $\mathcal{K}$ where each state $q \in Q$ is a valuation for a finite set of boolean variables $V = \{z_1, \dots, z_m\}$ such that $P \subseteq V$. The labeling function $\mathcal{L}$ is defined as the projection of 2^V to 2^P in the natural way. The transition relation is given by a BDD $T(V, V')$ over $V \cup V'$ where the set $V' = \{z' \mid z \in V\}$ of primed variables is used to define the value of the variables after the transition.

To efficiently compute $\mathcal{F}^{\mathcal{K}}_\phi$, we need a compact representation of $\succeq_{\mathsf{MC}}$-antichains. Under the hypothesis that the huge size of Q is the main obstacle, we consider a semi-symbolic representation of antichains, as a set of pairs $(B, \langle s, o\rangle)$ where B is a BDD over V. A pair $(B, \langle s, o\rangle)$ represents the set $[\![(B, \langle s, o\rangle)]\!] = \{(q, \langle s, o\rangle) \mid q \in [\![B]\!]\}$.

Let $L = \{(q_1, \langle s_1, o_1\rangle), (q_2, \langle s_2, o_2\rangle), \dots\}$ be an $\succeq_{\mathsf{MC}}$-antichain. Let $S_L = \{\langle s, o\rangle \mid \exists (q, \langle s, o\rangle) \in L\}$. We define $R(L) = \{(B, \langle s, o\rangle) \mid \langle s, o\rangle \in S_L \wedge [\![B]\!] = \{q \mid (q, \langle s, o\rangle) \in L\}$. It is easy to establish the following property of this encoding.

Lemma 4 *If L is an $\succeq_{\mathsf{MC}}$-antichain for all $(B_1, \langle s_1, o_1\rangle), (B_2, \langle s_2, o_2\rangle) \in R(L)$, if $\langle s_1, o_1\rangle \succ_{\mathsf{alt}} \langle s_2, o_2\rangle$, then $[\![B_1]\!] \cap [\![B_2]\!] = \varnothing$.*

We say that $R(L)$ is a *semi-symbolic* and *canonical* representation of $[\![L]\!]_{\succeq_{\mathsf{MC}}}$. The algorithm to compute $\mathsf{Post}_{\mathsf{MC}}(\cdot)$ follows the lines of Algorithm 1, using $2n$ boolean variables to encode a pair $\langle s, o\rangle$. The existential quantification over V is performed after synchronization over propositions P with the Kripke structure. Let $B_L(x, y, V)$ be the BDD that encodes with variables x, y the successors of $\langle s, o \rangle$ over a symbolic label encoded by variables V. We compute the BDD $C_L(x, y, V') = \exists V : B(V) \wedge T(V, V') \wedge B_L(x, y, V)$ and then we construct the encoding $R(\cdot)$ of its minimal elements.

6 Model-Checking: Performance Evaluation

Implementation. We have implemented the forward semi-symbolic model-checking algorithm using the same technology as for satisfiability (i.e., Python and PyCUDD). The sABW of the negation of the LTL formula is obtained as described in Section 3. We have interfaced our prototype with NuSMV in order to get the BDDs[9] obtained from models written in the SMV input language. This has two advantages. First, we can effectively run our algorithm on any available SMV model, making direct comparisons with NuSMV easy. Second, we are guaranteed to use *exactly* the same BDDs for the Kripke structure (with the same ordering on variables) than NuSMV, making comparisons with this tool very meaningful.

On the use of NuSMV. As for satisfiability, all our experiments were performed using NuSMV 2.4 without any option except "-dcx" which disables the creation of counter-examples. By default, NuSMV implements the following version of the LTL symbolic algorithm: it precomputes the reachable states of the Kripke structure and then evaluates a backward fixpoint expression (the Emerson-Lei algorithm) for checking emptiness of the product of the structure and the NBW of the formula (encoded with BDDs). Guiding the backward iterations with reachable states usually improves execution times dramatically. It also makes the comparison with our algorithm fair as it also only visits reachable states.

Benchmarks. We have compared our prototype with NuSMV on three families of scalable SMV models. The experiments were performed using the same environment as for satisfiability (see Section 4). Again, additional information about models and formulas can be found in[14].

The first family describes a gas station with an operator, one pump, and n customers (n is a parameter) waiting in line to use the pump. The operator manages the customer queue and activates or deactivates the pump. This resource-access protocol was used in [20] as an LTL model-checking benchmark. We have used the same LTL formulas as in [20]. The running times for n between 2 and 50 are given in Fig. 2(g). The difference in scalability is striking. While our tool is slower than NuSMV for n=2 (probably due to the overhead of using an interpreted language instead of C), it scales much better. For instance, for $n = 38$ NuSMV needs several minutes (between 233 and 418 seconds depending on the property), while our algorithm completes in just over 3 seconds for all properties. NuSMV is not able to verify models with 50 customers within 10 minutes while our algorithm handles them in less than 10 seconds.

The second family of models also comes from [20] and represents a stack, on which push, pop, empty and freeze operations can be performed. Each cell of the stack can

[9] These are essentially: the predicates appearing in the LTL formula, the initial constraints, the transition relation and the invariant constraints.

hold a value from the set $\{1,2\}$ and a freeze operation allows to permanently freeze the stack, after which the model runs a pointer along the stack from top to bottom repeatedly. At each step of this infinite loop, a "call" predicate indicates the value currently pointed.[10] As we needed a scalable set of formulas for at least one model to compare the scalability of our algorithm with NuSMV, we have provided and used our own specifications for this model. These specifications simply enforce that if the sequence of push operations "$12 \ldots n$" is performed and not followed by any pop until the freeze operation, then the subsequence of call operations "$n \ldots 21$" appears infinitely often.

Finally, the last family of models that we consider is a finite state version of the Lamport's bakery mutex protocol [13]. This protocol is interesting becauses it imposes fairness among all processes and again it is parametric in the number n of participating processes. Our model is large and grows very rapidly with the number of processes. For 2 processes, it uses 42 boolean variables and requires BDDs with a total of 7750 nodes to encode the model, for 4 processes, it uses 91 variables and BDDs with more than 20 million nodes. Again, our algorithm scales much better than the classical fully symbolic algorithm. For 3 processes, we are able to verify the fairness requirement in 730.6 seconds while NuSMV needs 28740.17s. Also, our algorithm requires much less memory than NuSMV, see Table 2(h) for the details.

7 Conclusion

In this paper, we have defined new algorithms for LTL satisfiability and model-checking. The new algorithms use a clever combination of the antichain method defined in [7] and BDDs. Our method differs fundamentally from the explicit and hybrid approach to LTL as it does not require the explicit construction of a NBW, and from the symbolic approach as it does not encode the NBW with BDDs.

With a prototype implementation written in Python, we outperform in time and memory usage the state-of-the-art implementation in NuSMV of the classical fully symbolic approach on all but one benchmark. More importantly, our implementation is able to handle LTL formulas and models that are too large for NuSMV.

There are several lines of future works to consider both on the theoretical side and on the practical side. First, we should investigate how we can take advantage of the structure of sABW that are produced from the LTL formula. Indeed, those sABW are weak in the sense of [17], a property that we do not exploit currently. Second, we use a notion of simulation which is called the direct simulation in the terminology of [10]. Weaker notions of simulation exist for NBW like the *fair simulation* or the *delayed simulation*. We should investigate their possible use instead of the direct simulation. This would allow for more pruning as antichains for those orders would be smaller. Third, high level heuristics should be investigated. Let us take an example. A pair of locations $\{l_1, l_2\}$ is an *incompatible* pair of locations in a sABW A if there is no word w such that w is accepted in A simultaneously from l_1 and l_2. In the forward satisfiability algorithm, it is easy to see that we can stop the exploration of any pairs $\langle s, o\rangle$ such that s contains an incompatible pair. We should look for easily (polynomial-time) checkable suffi-

[10] For example, if the stack contains, from bottom to top, $\{1,2\}$ then after the freeze operation, the model will behave like this : call2, call1, call2, call1, ...

cient conditions for incompatibility. Finally, a first release of our prototype (codenamed ALASKA) is available for download at http://www.ulb.ac.be/di/ssd/nmaquet/alaska/.

References

1. K. Baukus, S. Bensalem, Y. Lakhnech, and K. Stahl. Abstracting ws1s systems to verify parameterized networks. In *TACAS*, volume 1785 of *LNCS*, pages 188–203. Springer, 2000.
2. A. Cimatti, E. Clarke, F. Giunchiglia, and M. Roveri. Nusmv: A new symbolic model checker. *STTT*, 2(4):410–425, 2000.
3. E. Clarke, O. Grumberg, and K. Hamaguchi. Another look at LTL model checking. In *CAV*, volume 818 of *LNCS*, pages 415–427. Springer, 1994.
4. M. Daniele, F. Giunchiglia, and M. Vardi. Improved automata generation for linear temporal logic. In *CAV*, volume 1633 of *LNCS*, pages 249–260. Springer, 1999.
5. L. de Alfaro, T.A. Henzinger, and R. Majumdar. From verification to control: Dynamic programs for omega-regular objectives. In *LICS*, pages 279–290. IEEE, 2001.
6. M. De Wulf, L. Doyen, T.A. Henzinger, and J-F. Raskin. Antichains: A new algorithm for checking universality of finite automata. In *CAV*, LNCS 4144, pages 17–30. Springer, 2006.
7. L. Doyen and J-F. Raskin. Improved algorithms for the automata-based approach to model-checking. In *TACAS*, volume 4424 of *LNCS*, pages 451–465. Springer-Verlag, 2007.
8. C. Fritz. Constructing Büchi automata from LTL using simulation relations for alternating Büchi automata. In *CIAA*, volume 2759 of *LNCS*, pages 35–48. Springer, 2003.
9. P. Gastin and D. Oddoux. Fast LTL to Büchi automata translation. In *CAV*, volume 2102 of *Lecture Notes in Computer Science*, pages 53–65. Springer-Verlag, 2001.
10. S. Gurumurthy, O. Kupferman, F. Somenzi, and M. Vardi. On complementing nondeterministic Büchi automata. In *CHARME*, volume 2860 of *LNCS*, pages 96–110. Springer, 2003.
11. A. Harding. *Symbolic Strategy Synthesis For Games With LTL Winning Conditions*. PhD thesis, University of Birmingham, 2005.
12. T.A. Henzinger, O. Kupferman, and S. Qadeer. From *pre*historic to *post*modern symbolic model checking. In *CAV*, volume 1427 of *LNCS*, pages 195–206. Springer, 1998.
13. Leslie Lamport. A new solution of dijkstra's concurrent programming problem. *ACM*, 17(8):453–455, 1974.
14. N. Maquet M. De Wulf, L. Doyen and J.-F. Raskin. Antichains: Alternative algorithms for LTL satisfiability and model-checking. Technical Report 2008.84, CFV, Belgium.
15. K. L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1993.
16. S. Miyano and T. Hayashi. Alternating finite automata on omega-words. In *CAAP*, pages 195–210, 1984.
17. S. Rohde. *Alternating Automata and the Temporal Logic of Ordinals*. PhD thesis, University of Illinois at Urbana-Champaign, 1997.
18. K Rozier and M. Vardi. Ltl satisfiability checking. In *14th International SPIN Workshop*, volume 4595 of *LNCS*, pages 149–167. Springer, 2007.
19. T. Ruys and G. Holzmann. Advanced Spin tutorial. In *SPIN*, volume 2989 of *LNCS*, pages 304–305. Springer, 2004.
20. R. Sebastiani, S. Tonetta, and M. Vardi. Symbolic systems, explicit properties: On hybrid approaches for LTL symbolic model checking. In *CAV*, LNCS 3576, pages 350–363, 2005.
21. F. Somenzi. CUDD: CU Decision Diagram Package. University of Colorado, 1998.
22. F. Somenzi and R. Bloem. Efficient Büchi automata from LTL formulae. In *CAV*, volume 1855 of *LNCS*, pages 248–263. Springer, 2000.
23. M. Vardi. An automata-theoretic approach to linear temporal logic. In *8th Banff Higher Order Workshop*, volume 1043 of *LNCS*, pages 238–266. Springer, 1995.
24. M. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, 1994.