Simplifying LTL Model Checking Given Prior Knowledge

Alexandre Duret-Lutz[3], Denis Poitrenaud[1,2], Yann Thierry-Mieg✉[1]

[1] Sorbonne Université, CNRS, LIP6, F-75005 Paris, France
denis.poitrenaud@lip6.fr, yann.thierry-mieg@lip6.fr
[2] Université Paris Cité, F-75006 Paris, France
[3] EPITA, LRE, Le Kremlin-Bicêtre, France
adl@lrde.epita.fr

Abstract. We consider the problem of the verification of an LTL specification φ on a system S given some prior knowledge K, an LTL formula that S is known to satisfy. The automata-theoretic approach to LTL model checking is implemented as an emptiness check of the product $S \otimes A_{\neg\varphi}$ where $A_{\neg\varphi}$ is an automaton for the negation of the property. We propose new operations that simplify an automaton $A_{\neg\varphi}$ *given* some knowledge automaton A_K, to produce an automaton B that can be used instead of $A_{\neg\varphi}$ for more efficient model checking.
Our evaluation of these operations on a large benchmark derived from the MCC'22 competition shows that even with simple knowledge, half of the problems can be definitely answered without running an LTL model checker, and the remaining problems can be simplified significantly.

1 Introduction — Knowledge is Power

LTL model checking consists in verifying whether all infinite executions of a system S satisfy an LTL formula φ, *i.e.*, $\mathscr{L}(S) \subseteq \mathscr{L}(\varphi)$. In this case we write $S \models \varphi$. In the automata-theoretic approach to model checking [47], this inclusion test is usually implemented as an emptiness check of the product of two automata: $\mathscr{L}(S \otimes A_{\neg\varphi}) = \emptyset$, where $A_{\neg\varphi}$ represents the negation of φ.

The premise of this paper is that we assume to have some additional knowledge K about S. In particular, the knowledge we consider are over-approximations of the system: $\mathscr{L}(S) \subseteq \mathscr{L}(K)$. For instance K might be an LTL formula that has already been proven on S. Of course if K implies φ, *i.e.* $\mathscr{L}(K) \subseteq \mathscr{L}(\varphi)$, then φ holds as well since $\mathscr{L}(S) \subseteq \mathscr{L}(K)$. And if $\mathscr{L}(K) \subseteq \mathscr{L}(\neg\varphi)$, any run of the system is a counter-example.

But if none of these basic implications hold, we can still benefit from prior knowledge. We show that verifying $S \models \varphi$ *given* K is equivalent to checking $\mathscr{L}(S \otimes B) = \emptyset$ for an automaton B that is *simpler* than $A_{\neg\varphi}$, hopefully allowing a faster exploration of $S \otimes B$.

As an example the automaton $A_{\neg\varphi}$ that is on the left of Figure 3 (page 8) can be replaced by the automaton B that is on the right of the same figure. This new automaton is smaller, uses fewer atomic propositions, is now deterministic, and needs fewer acceptance sets because it is now a terminal automaton [9,28]. Using this automaton B should therefore simplify the job of a model checker.

This paper is organized as follows. In Section 2 we formalize notion of $S \models \varphi$ *given* K from the point of view of languages, and discuss possible goals when transposing this on automata. In Section 3 we pose useful definitions, then Section 4 proposes basic and Section 5 advanced automata operations that aim to simplify $A_{\neg\varphi}$ based on some given knowledge K. In Section 6 we propose costlier automata operations that aim to modify $A_{\neg\varphi}$ to make it stutter-insensitive, within the bounds allowed by some knowledge K. Finally, in Section 8 we evaluate the above techniques on a large third-party benchmark provided by the model checking contest [26].

2 Bounding Languages "Given That..."

In this section, we focus on providing justification for our approach at the *language* level. The language $\mathscr{L}(\mathrm{X})$ of a system or property X is a set of infinite words over an alphabet Σ, $\mathscr{L}(X) \subseteq \Sigma^\omega$. We denote $\overline{\mathscr{L}(X)} = \Sigma^\omega \setminus \mathscr{L}(X)$ the complement of the language of X.

A system S satisfies property φ, denoted $S \models \varphi$ if and only if the language $\mathscr{L}(S)$ of the system is a subset of the property language $\mathscr{L}(\varphi)$, *i.e.*, $\mathscr{L}(S) \subseteq \mathscr{L}(\varphi)$. When φ is an LTL formula, the classical automaton-based approach [47] is to test $\mathscr{L}(S) \cap \mathscr{L}(\neg\varphi) = \emptyset$, *i.e.*, perform an emptiness check with the language of the negated property.

In the following, we assume that $\mathscr{L}(S) \neq \emptyset$ since the empty system would satisfy any property and its negation.

Now, consider a property K (a *knowledge*) such that it has already been established that $S \models K$, *i.e.*, we know that $\mathscr{L}(S) \subseteq \mathscr{L}(K)$. This *a priori* knowledge gives us some degrees of freedom when testing whether S satisfies a new property φ. Indeed, we already know that words outside $\mathscr{L}(K)$ are definitely *not* part of $\mathscr{L}(S)$.

The main intuition is given by Fig. 1. Since $\mathscr{L}(S) \subseteq \mathscr{L}(K)$, it is safe to replace the test $\mathscr{L}(S) \cap \mathscr{L}(\neg\varphi) = \emptyset$ by a test $\mathscr{L}(S) \cap \mathscr{L}(B) = \emptyset$ where $\mathscr{L}(B)$ is built from $\mathscr{L}(\neg\varphi)$ by either removing or including words of $\overline{\mathscr{L}(K)}$. Indeed, words in $\overline{\mathscr{L}(K)}$ are not part of the system, so they cannot belong to $\mathscr{L}(S) \cap \mathscr{L}(\neg\varphi)$.

This leads to the following theorem whose proof follows immediately from Figure 1.

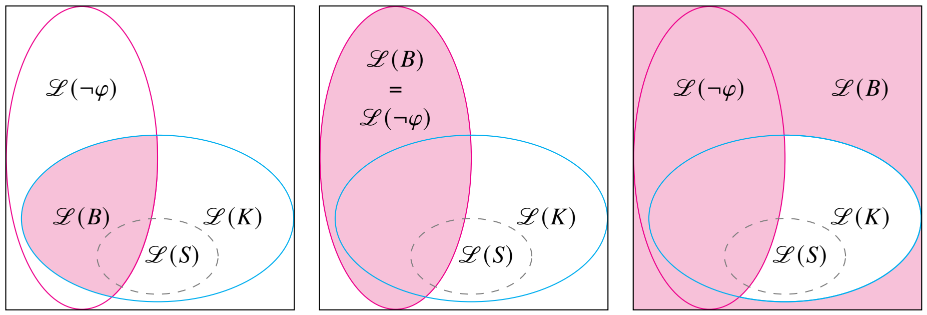
Theorem 1. *Let S be a system, and K a property such that $\mathscr{L}(S) \subseteq \mathscr{L}(K)$. For any property φ, we can define a $\mathscr{L}(B)$ such that $\mathscr{L}(S) \cap \mathscr{L}(\neg\varphi) = \emptyset$ if and only if $\mathscr{L}(S) \cap \mathscr{L}(B) = \emptyset$, by choosing $\mathscr{L}(B)$ between the following bounds:*

$$\mathscr{L}(\neg\varphi) \cap \mathscr{L}(K) \quad \subseteq \quad \mathscr{L}(B) \quad \subseteq \quad \mathscr{L}(\neg\varphi) \cup \overline{\mathscr{L}(K)}$$

The lower bound $\mathscr{L}(\neg\varphi) \cap \mathscr{L}(K)$ is called the *restriction* of $\neg\varphi$: it is constructed from $\mathscr{L}(\neg\varphi)$ by removing words from $\overline{\mathscr{L}(K)}$ (Fig. 1a). The upper bound $\mathscr{L}(\neg\varphi) \cup \overline{\mathscr{L}(K)}$ is the *relaxation* of $\neg\varphi$, constructed from $\mathscr{L}(\neg\varphi)$ by adding words from $\overline{\mathscr{L}(K)}$ (Fig. 1c).

The above theorem gives us more freedom in the automata-theoretic approach to LTL model checking. In this context, both the property φ and the knowledge K are expressed as linear-time temporal logic (LTL) formulas which can be converted into automata over infinite words.

Thus, model checking $S \models \varphi$ is implemented as $\mathscr{L}(S \otimes A_{\neg\varphi}) = \emptyset$ where $A_{\neg\varphi}$ is an automaton for $\neg\varphi$, and $\otimes$ is the product of automata [47]. Here, we want to use



(a) The most restricted $\mathscr{L}(B)$ that can be constructed from K, $\mathscr{L}(B) = \mathscr{L}(\neg\varphi) \cap \mathscr{L}(K)$

(b) Classic approach simply using the language of $\neg\varphi$, $\mathscr{L}(B) = \mathscr{L}(\neg\varphi)$.

(c) The most relaxed $\mathscr{L}(B)$ that can be constructed given K, $\mathscr{L}(B) = \mathscr{L}(\neg\varphi) \cup \overline{\mathscr{L}(K)}$.

Fig. 1: The outside box represents all words in Σ^ω. Each language is depicted as an ellipse, with the language of the system $\mathscr{L}(S)$ inside the knowledge $\mathscr{L}(K)$ but we do not know whether the system language overlaps the negated property language $\mathscr{L}(\neg\varphi)$. The language $\mathscr{L}(B)$ represented in magenta can be chosen anywhere between these extremes to replace $\mathscr{L}(\neg\varphi)$ in the model-checking procedure.

Theorem 1 to find a *simpler* automaton B such that model checking with $\mathscr{L}(S \otimes B) = \emptyset$ is more efficient. Contrary to intuition, choosing the automaton B with the smallest *language* might be counter-productive, because a small language does not necessarily equate to a small automaton.

To make model checking more efficient, we target the following goals:

smaller or more deterministic Reducing the size of B, or making it more deterministic can often reduce the size of the product $S \otimes B$. (Blahoudek et al. [4] suggest that contrary to previous measurements [39], “smaller” is more important than “more deterministic” for model checking.)

simpler strength class The emptiness check algorithms can be simplified if B belongs to simpler classes of automata, such as *weak*, or *terminal* automata [6,9,35].

stutter-insensitive For concurrent systems, many partial-order reductions (POR) techniques [32,46,21] and structural reductions [22,44] can be used when it is known that B is stutter-insensitive.

fewer atomic proposition checks Reducing the number of atomic propositions and the syntactic complexity of the formulas labeling the edges of B can reduce the time required to build $S \otimes B$ in explicit model checking [5], and reducing the set of observed propositions also helps the aforementioned POR based techniques.

The techniques we will propose mainly attempt to reduce the size of the automata, their number of atomic propositions, and attempt to make them stutter-insensitive. Any determinism improvement or strength reduction is a welcome side effect.

3 Simplifying Automata "Given That..."

We now turn the language bounds of Section 2 into automata constructions. We use a variant of Büchi automata called *transition-based generalized Büchi automata* (TGBA). This variant uses accepting transitions instead of accepting states. Additionally, the acceptance condition is generalized: a run has to visit *multiple* accepting sets of transitions infinitely often. This variant is particularly compact to express weak fairness conditions [11], and it also makes our subsequent definitions easier without loss of generality.

3.1 Definitions

The following definitions are freely adapted from the literature.

Let $\mathbb{B} = \{\bot, \top\}$ represent the Boolean set, and let AP represent a set of Boolean atomic propositions. A valuation ℓ is a function from AP to $\mathbb{B}$. The set of valuations is denoted $\mathbb{B}^{AP}$. The set of Boolean formulas over AP is denoted $\mathbb{B}(AP)$. In the following, we consider words that are infinite sequences of valuations, therefore the alphabet Σ of Section 2 is $\Sigma = \mathbb{B}^{AP}$. For an atomic proposition $a \in AP$ we use $\bar{a}$ or $\neg a$ interchangeably to represent its negation.

Definition 1 (TGBA). *A Transition-based Generalized Büchi Automaton (TGBA), is a structure $A = \langle AP, Q, \iota, Acc, \delta\rangle$ where*

- *AP is a finite set of Boolean atomic propositions,*
- *Q is a finite set of states,*
- *$\iota \in Q$ is the initial state,*
- *Acc is a finite set of acceptance marks (denoted ⓿, ❶, ❷, etc.)*
- *$\delta \subseteq Q \times \mathbb{B}(AP) \times 2^{Acc} \times Q$ is the transition relation where we use $t = q \xrightarrow{f,a} q'$ to denote an element $t \in \delta$, f is a Boolean formula over AP that we call the* label *of the transition and a is a set of acceptance marks.*

A run *of A on an infinite word $w = \ell_1\ell_2\ell_3 \ldots \in (\mathbb{B}^{AP})^\omega$ is an infinite sequence of connected transitions $\rho = q_1 \xrightarrow{f_1,a_1} q_2 \xrightarrow{f_2,a_2} q_3 \xrightarrow{f_3,a_3} q_4 \ldots \in \delta^\omega$ such that $q_1 = \iota$ and for all i, $\ell_i \Rightarrow f_i$. (Recall that ℓ_i is a valuation of all atomic propositions, therefore a conjunction of atomic propositions, in negative or positive form, but f_i is a Boolean formula.) A run is* accepting *iff for each mark $m \in Acc$ there are infinitely many i such that $m \in a_i$.*

The language *of A, denoted $\mathscr{L}(A)$, is the set of all infinite words w such that there exists an accepting run of A on w.*

Theorem 2 (TGBA for a formula [11,19,20]). *Given an LTL formula φ over AP, one can build a TGBA A_φ with $O(2^{|\varphi|})$ states such that $\mathscr{L}(A_\varphi) = \mathscr{L}(\varphi)$.*

For instance the leftmost automaton of Figure 3 (page 8) is a TGBA for $\mathsf{F}(p \wedge r) \vee \mathsf{G}((\mathsf{F}q) \vee (\mathsf{F}\bar{q}))$. An accepting run has to encounter marks ⓿ and ❶ infinitely often. Therefore, any run reaching state 1 is accepting, and any run reaching state 2 is accepting if both q and $\bar{q}$ hold infinitely often. Note that A_φ is not unique. There is a vast literature on techniques for building and simplifying automata [15,42,16,18,2,13, ...].

An obvious optimization is to discard the useless parts of the automaton by trimming it. The *trim* of an automaton A, denoted $\mathit{Trim}(A)$, is the restriction of A to the transitions and states that appear in at least one accepting run of A. Doing so preserves the language of A. This operation can be done in linear time by studying the strongly connected components of the automaton [15].

The intersection of the languages of two automata A_1 and A_2 is represented by a product $A_1 \otimes A_2$ such that $\mathscr{L}(A_1 \otimes A_2) = \mathscr{L}(A_1) \cap \mathscr{L}(A_2)$.

Definition 2 (Product of TGBA). *Given two automata* $A_1 = \langle AP_1, Q_1, \iota_1, Acc_1, \delta_1\rangle$ *and* $A_2 = \langle AP_2, Q_2, \iota_2, Acc_2, \delta_2\rangle$*, where* $Acc_1 \cap Acc_2 = \emptyset$*, the product* $A_1 \otimes A_2$ *is the automaton* $\langle AP, Q, \iota, Acc, \delta\rangle$ *where:*

- $AP = AP_1 \cup AP_2$
- $Q = Q_1 \times Q_2$
- $\iota = (\iota_1, \iota_2)$
- $Acc = Acc_1 \cup Acc_2$.
- $\delta = \left\{ (q_1, q_2) \xrightarrow{f_1 \wedge f_2, a_1 \cup a_2} (q_1', q_2') \;\middle|\; q_1 \xrightarrow{f_1, a_1} q_1' \in \delta_1, q_2 \xrightarrow{f_2, a_2} q_2' \in \delta_2 \right\}$

For instance Figure 2 shows in the bottom right the product $A_{\neg\varphi} \otimes A_K$ of the two surrounding automata. The transitions that would be removed by *Trim* are dashed.

One can also define the sum of two TGBA $A_1 \oplus A_2$ such that $\mathscr{L}(A_1 \oplus A_2) = \mathscr{L}(A_1) \cup \mathscr{L}(A_2)$, and the complement $\overline{A}$ such that $\mathscr{L}(\overline{A}) = (\mathbb{B}^{AP})^\omega \setminus \mathscr{L}(A)$. We omit the precise definition of these operations. While sum and product are cheap operations (at most quadratic in the size of the automata), the complement is worse than exponential [48,38] so it is often desirable to avoid it (e.g., we prefer to compute $A_{\neg\varphi}$ instead of $\overline{A_\varphi}$).

4 Basic Strategies

The simplest way to apply Theorem 1 is to build automata for the most restricted and the most relaxed languages pictured in Figure 1. Consider the following two definitions:

$$\min\nolimits_{|K}(A_{\neg\varphi}) = A_{\neg\varphi} \otimes A_K \qquad (1)$$

$$\max\nolimits_{|K}(A_{\neg\varphi}) = A_{\neg\varphi} \oplus A_{\neg K} \qquad (2)$$

When B is chosen as $\min_{|K}(A_{\neg\varphi})$, we are using the most restricted language of Figure 1a. If B is $\max_{|K}(A_{\neg\varphi})$ we are using the most relaxed language of Figure 1c.

Note that if $\mathscr{L}(\min_{|K}(A_{\neg\varphi})) = \emptyset$, then it follows from the definition that $\mathscr{L}(K) \subseteq \mathscr{L}(\varphi)$, and since $\mathscr{L}(S) \subseteq \mathscr{L}(K)$ we have $S \models \varphi$. Dually, if $\mathscr{L}(\max_{|K}(A_{\neg\varphi})) = (\mathbb{B}^{AP})^\omega$, then $\mathscr{L}(K) \subseteq (\neg\varphi)$, which means that $S \models \neg\varphi$ (i.e., every run of S is a counterexample of φ) and therefore $S \not\models \varphi$ (because S is nonempty).

While the emptiness check of a TGBA $\mathscr{L}(A) = \emptyset$ can be performed in linear time [11], the universality test $\mathscr{L}(A) = (\mathbb{B}^{AP})^\omega$ requires exponential time [17]. Fortunately the universality test $\mathscr{L}(\max_{|K}(A_{\neg\varphi})) = (\mathbb{B}^{AP})^\omega$ can be avoided by replacing it with $\mathscr{L}(\min_{|K}(A_\varphi)) = \emptyset$ provided a formula for φ is known.

Moreover, the automata products and sums in the above $\min_{|K}$ and $\max_{|K}$ constructions can also be replaced by logical operations on formulas before translating them to TGBA, as in $A_{\neg\varphi\wedge K}$ and $A_{\neg\varphi\vee\neg K}$ respectively.

In the case where the min and max automata are neither empty nor universal, their sizes are unlikely to be smaller than the original $A_{\neg\varphi}$. In a way, using these automata for model checking is similar to asking the model checker to prove K in addition to $\neg\varphi$. As stated in Section 2, we would prefer to select a B that is "simpler" than $A_{\neg\varphi}$.

The knowledge K could contain atomic propositions that do not appear in $\neg\varphi$. Let P be the set of atomic propositions that appear in K but not in φ. To avoid introducing needless atomic propositions in P, we suggest to existentially quantify them. This quantification can be done precisely on the automaton A_K by existentially quantifying P from all labels, or it can be *over approximated* on the LTL formula K by quantifying P from all its Boolean subformulas (considered individually). We note $QE(P,K)$ the latter operation. To show that this is an over-approximation, consider the unsatisfiable formula $K = \mathsf{X}(a \wedge b) \wedge \mathsf{X}(\bar{a} \wedge b)$ and $P = \{a\}$. We have $(\exists a,\ a \wedge b) = b$ and $(\exists a,\ \bar{a} \wedge b) = b$, therefore, $QE(P,K) = \mathsf{X}(b) \wedge \mathsf{X}(b) = \mathsf{X}(b)$ which is satisfiable.

Assuming P contains the atomic propositions of K that are not in φ, let us introduce the following notations:

$$\min{}^{\exists}_{|K}(\neg\varphi) = A_{(\neg\varphi)\wedge QE(P,K)} \tag{3}$$

$$\max{}^{\exists}_{|K}(\neg\varphi) = A_{(\neg\varphi)\vee\neg QE(P,K)} \tag{4}$$

5 Using Transition-Based Boolean Bounds on Labels

In this section, we investigate how to leverage theorem 1 so that given an automaton for a knowledge K, we rewrite the automaton $A_{\neg\varphi}$ into a simpler automaton B.

Simplicity here is measured syntactically on the automaton; we want an automaton that has fewer states, fewer transitions, fewer atomic propositions, fewer acceptance marks, and simpler (smaller) Boolean formulas labeling the transitions of the automaton.

To achieve this, we propose to compute a set of Boolean bounds for each transition of the automaton $A_{\neg\varphi}$. These bounds enable more flexibility in the selection of transition labels by providing the most restrictive and the most relaxed Boolean formulas that can label each transition.

Minato's algorithm [30] is a recursive way to rewrite a Boolean formula as a prime-irredundant cover, which is very compact in general. The algorithm works recursively using formulas in three-valued logic, and Minato [30, Section 4.4] suggests an implementation of this algorithm using Binary Decision Diagrams [8] where a three-valued formula is simply bounded using two Boolean functions: (f_{low}, f_{high}) and the algorithm generates an irredundant sum-of-product f' such that $f_{low} \Rightarrow f' \Rightarrow f_{high}$. In other words, f' is generated as a disjunction of conjunctions of literals, such that no conjunct is uncessary, and no literal can be removed from any conjunct. We use this algorithm to simplify transition labels, as it removes literals that are unnecessary to stay within those bounds.

In Sections 5.1 and 5.2, we introduce strategies to compute Boolean lower and upper bounds for each label of the automaton. Then, in Section 5.3, we show how to simplify transition labels by using Minato's algorithm on the computed bounds. This approach preserves the transition structure of the automaton. It can sometimes remove transitions (if its label becomes $\bot$), it can remove states (when they become unreachable), it can reduce the number of atomic propositions used, and it generally simplifies the expression of the labels. So, contrary to the $\min^{\exists}_{|K}(\neg\varphi)$ and $\max^{\exists}_{|K}(\neg\varphi)$ approaches presented in Section 4, this approach always produces a simpler automaton.

5.1 Boolean upper bounds

The first step consists in realizing that since $S \models K$, in *every state* of A_K we are over-approximating the state the system S might be in. Some paths in A_K might not be realizable by S, but the system definitely cannot do anything that K does not allow.

We start by building the synchronized product $A_{\neg\varphi} \otimes A_K$ in which every state is a pair (q, k). We can then apply the *Trim* operation to discard any transition that does not belong to an accepting Strongly Connected Component (SCC) or to the prefix of one, and then discard any state of the product unreachable from the initial state.

Now consider for a given state q of $A_{\neg\varphi}$ the set of states Q_q of the knowledge automaton A_K in correspondence with q. The state of the system S in this set of states can be over-approximated as the logical disjunction of the formulas labeling any transition that is outgoing from any state in Q_q.

Definition 3 (Knowledge-based state guarantee). *Given two automata $A_{\neg\varphi} = \langle AP, Q, \iota, Acc, \delta\rangle$, and $A_K = \langle AP, Q_K, \iota_K, \delta_K, Acc_K\rangle$, let $Trim(A_{\neg\varphi} \otimes A_K) = \langle AP, Q_P, \iota_P, Acc_P, \delta_P\rangle$ be the trim product of $A_{\neg\varphi}$ and A_K.*

For any state $q \in Q$, let $Q_q \subseteq Q_K$ denotes the subset of states of A_K that can synchronize with q in the product:

$$Q_q = \{k \in Q_K \mid (q, k) \in Q_P\}$$

From this set of states, we define the state guarantee *in state q :*

$$\mathsf{SG}(q) = \bigvee_{k \in Q_q} \; \bigvee_{k \xrightarrow{f,a} k' \in \delta_K} f$$

that represents the disjunction of all transition labels of A_K that leave a state of Q_q.

In the trim product $Trim(A_{\neg\varphi} \otimes A_K)$, consider a transition $(q, k) \xrightarrow{f \wedge f_k, a \cup a_k} (q', k')$ that was built as a product of $q \xrightarrow{f,a} q'$ and $k \xrightarrow{f_k, a_k} k'$. Then it is guaranteed that $f_k \Rightarrow \mathsf{SG}(q)$ by construction. Hence, when the component $A_{\neg\varphi}$ of the product is known to be in q, this $\mathsf{SG}(q)$ is an over approximation of the labels f_k that the transitions in component A_K can satisfy.

Since A_K overapproximates the system S, it is also true that $\mathsf{SG}(q)$ will overapproximate the behaviors of the states of S that can synchronize with q. This state guarantee formula thus provides an upper bound or over approximation of the system state when reaching q; therefore, for any transition $q \xrightarrow{f,a} q' \in \delta$, *relaxing* the transition label f to accept $f \vee \neg\mathsf{SG}(q)$ would not modify the language of the product with the system S, since the system cannot satisfy $\neg\mathsf{SG}(q)$ in this state of the product.

Figure 2 shows examples of computation of SG.

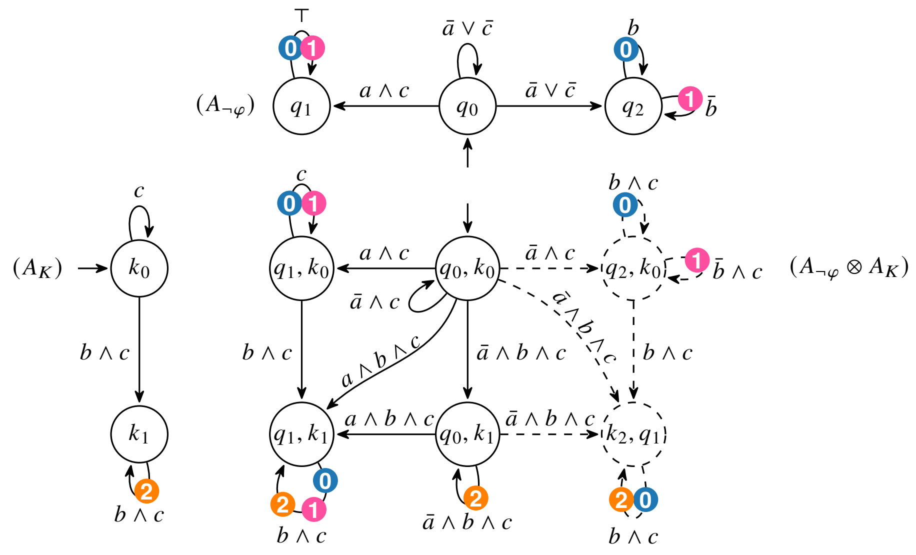


Fig. 2: Example of product of $A_{\neg\varphi} \otimes A_K$ for $\neg\varphi = \mathsf{F}(a \wedge c) \vee \mathsf{G}((\mathsf{F}b) \wedge (\mathsf{F}\bar{b}))$ and $K = \mathsf{FG}(b) \wedge \mathsf{G}(c)$. The dashed transitions are those removed by *Trim*. We have $\mathsf{SG}(q_0) = \mathsf{SG}(q_1) = (c) \vee (b \wedge c) \vee (b \wedge c) = c$ because these two states can be synchronized with all the states of A_K, therefore their state guarantee is the disjunction of all labels of A_K. This result indicates that when the system is synchronized with state q_0, it will always satisfy c. We have $\mathsf{TG}(q_1 \xrightarrow{\top,\{0,1\}} q_1) = (c) \vee (b \wedge c) = c$, which indicates that when a transition of the system is synchronized with this self-loop, it will always satisfy c. Finally, $\mathsf{TG}(q_0 \xrightarrow{\bar{a}\vee\bar{c},\emptyset} q_2) = \bot$ because the only transition synchronizing with this one was trimmed, showing that this transition is not needed.

```
A = spot.translate('F(a & c) | G(Fb & F!b)')
K = 'FG(b) & G(c)'
Ab = spot.update_bounds_given(A, K)
Aminato = spot.bounds_simplify(Ab)
Asimpl = Aminato.postprocess('small')
display_inline(A, Ab, Aminato, Asimpl)
```

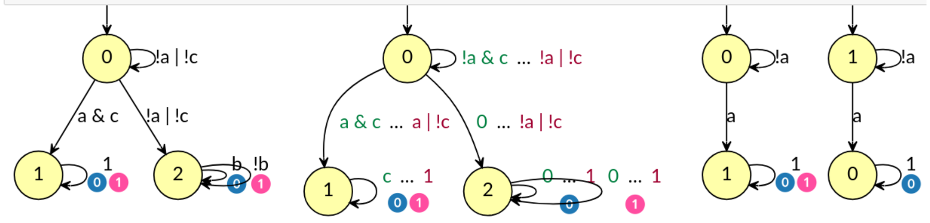


Fig. 3: Use of Spot in a Jupyter notebook to integrate some knowledge $K = \mathsf{FG}(b) \wedge \mathsf{G}(c)$ into the automaton for $\neg\varphi = \mathsf{F}(a \wedge c) \vee \mathsf{G}((\mathsf{F}b) \wedge (\mathsf{F}\bar{b}))$. Note that this is the same example as Figure 2 where the construction of the bounds is explained in detail. $A_{\neg\varphi}$ is on the left. The integration of knowledge K is represented as an intermediate "bounded automaton" in which each transition is bounded according to Theorem 3 (second automaton). Applying Minato's algorithm gives the third automaton, which can be further simplified to the rightmost automaton reducing the problem to verification of $\mathsf{F}a$. Note that Spot's notations differ slightly from those used in the paper, for instance `0`, `1`, and `!a|!c` stand for $\bot$, $\top$ and $\bar{a} \vee \bar{c}$ respectively.

5.2 Boolean lower bounds

Let us look at a way to restrict a transition label of $A_{\neg\varphi}$ without limiting the ways in which the system can synchronize with this transition. For this purpose, we introduce $\mathsf{TG}(t)$ the *transition guarantee* of a transition $t = q \xrightarrow{f,a} q'$ of $A_{\neg\varphi}$, as the disjunction of all labels of transitions of K that synchronize with t in the trim product.

Definition 4 (Knowledge-based transition guarantee). *Using the same automata as in Definition 3, for any transition $t = q \xrightarrow{f,a} q' \in \delta$, we consider the set of formulas $K_t = \{f_k \mid (q,k) \xrightarrow{f\wedge f_k, a\cup a_k} (q',k') \in \delta_P\}$ that appear on transitions of A_K that synchronize with t in the product. From the disjunction of this set of formulas, we define the* transition guarantee *for transition t:*

$$\mathsf{TG}(t) = \bigvee_{f_k \in K_t} f_k$$

Intuitively the label f of t can be restricted to $f' = f \wedge \mathsf{TG}(t)$ since this formula is already enough to match all labels of transitions of K that would synchronize with t in an accepted run. Hence, labeling t with f' is also enough to match all states of the system S that would synchronize with t with its original label f.

Figure 2 shows examples of computation of TG.

5.3 Using the bounds

Theorem 3. *Using the same automata as in Definition 3, consider a transition $t = q \xrightarrow{f,a} q' \in \delta$, and let $B = \langle AP, Q, \iota, Acc, \delta \setminus \{t\} \cup \{t'\}\rangle$ be a copy of $A_{\neg\varphi}$ where t has been replaced by $t' = q \xrightarrow{f',a} q'$ where $f' \in \mathbb{B}(AP)$ is any formula such that*

$$\underbrace{f \wedge \mathsf{TG}(t)}_{\textit{lower bound}} \quad \Rightarrow \quad f' \quad \Rightarrow \quad \underbrace{f \vee \neg\mathsf{SG}(q)}_{\textit{upper bound}}$$

Then $\mathscr{L}(B \otimes A_K) = \mathscr{L}(A_{\neg\varphi} \otimes A_K)$

For size reasons, the proof is in Appendix A.

Our implementation of this construction is an extension of Spot [14] in which the Boolean bounds of Theorem 3 can be represented directly on the automaton. Figure 3 shows our implementation at work. Our knowledge bound integration function `spot.update_bounds_given` can be called repeatedly to integrate multiple knowledge incrementally, as we will discuss in Section 7.

To select a simple label compatible with the bounds, we apply Minato's algorithm [30] (introduced at the top of Section 5) to compute a simpler label f' such that $f \wedge \mathsf{TG}(t) \Rightarrow f' \Rightarrow f \vee \neg\mathsf{SG}(q)$. Note that when the lower bound is $\bot$, Minato's algorithm will always return $f' = \bot$ (the transition can be removed), else if the upper bound is $\top$, $f' = \top$ will be returned.

If A and K are defined over different sets of atomic propositions, the result of Theorem 3 might include atomic propositions from K that were not in A, which is

```
A = spot.translate('XFa'); K = spot.translate('!a', 'Buchi')
sirelax = spot.stutterize_given(A, [K], True).postprocess('small')
sirestrict = spot.stutterize_given(A, [K], False).postprocess('small')
display_inline(A, K, sirestrict, sirelax, per_row=2)
```

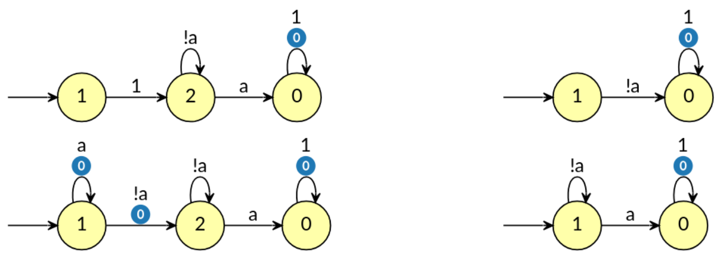


Fig. 4: Use of Spot in a Jupyter notebook to integrate some trivial knowledge $K = \bar{a}$ (top right automaton) into the stutter-sensitive automaton for $\neg\varphi = \mathsf{XF}(a)$ (top-left automaton) and turn it into a stutter-insensitive automata. Simplified automata for $\mathit{sirestrict}_{|K}(A)$ and $\mathit{sirelax}_{|K}(A)$ are given on the bottom left and right respectively.

counterproductive. In this case, we simplify K by existential quantification of the propositions that are not in A to produce an automaton K_{QE}. The language of this automaton contains $\mathscr{L}(K)$, therefore it also contains $\mathscr{L}(S)$ and it can still be used as a knowledge. We denote $BM_{|K}(A)$ the "Bounded by Minato" automaton, i.e., the automaton built from A and K by applying Minato's algorithm on the Boolean bounds computed by Theorem 3 with existential quantification of the atomic propositions not in A.

6 Building Stutter-Insensitive Automata "Given that..."

When model checking a concurrent system S against a formula φ, several advanced and very effective simplification techniques can be used when it is known that $\mathscr{L}(A_{\neg\varphi})$ is stutter-insensitive with reductions up to a factorial factor [32,46,21,22,44].

In this section, we consider the case where the language of $A_{\neg\varphi}$ is *stutter-sensitive*, and, given a knowledge K, we want to replace $A_{\neg\varphi}$ by an automaton B whose language is *stutter-insensitive*. Even if B is "bigger" than $A_{\neg\varphi}$, model checking might become more efficient thanks to the aforementioned simplifications.

Given a word w, let $[w]$ be the set of stutter-equivalent words that can be obtained from w by finitely duplicating letters or removing repetitions.[4] A language $\mathscr{L}(A)$ is stutter-sensitive iff there exists at least one equivalence class $[w]$ that is *only partly* covered by $\mathscr{L}(A)$, i.e., such that $[w] \cap \mathscr{L}(A) \neq \emptyset$ and $[w] \cap \overline{\mathscr{L}(A)} \neq \emptyset$.

As an example, consider the automaton $A_{\mathsf{XF}a}$ *given that* $K = \bar{a}$. Figure 4 shows this example using automata. The knowledge provided here is very simple, but this kind of effect could occur anywhere in the automaton, not just starting in the initial state. Automaton $A_{\mathsf{XF}a}$ is stutter-sensitive: it rejects the word $w = a\bar{a}\bar{a}\bar{a}\cdots$, but it accepts words starting with more than one a. However, notice w is outside $\mathscr{L}(K)$, so by Theorem 1, w could be added to the language of B to make it stutter-insensitive

[4] $[w]$ is an equivalence class for the $\sim^{\lim}$ relation of Peled et al. [33, Lemma 3].

(now accepting $\mathsf{F}a$), giving the bottom-right automaton of Figure 4. Another strategy would be to remove all words in $[w]$ from B: since all these words all start by a, the whole class $[w]$ is outside $\mathscr{L}(K)$. This second option, corresponding to the formula $\mathsf{G}(a) \vee \mathsf{F}(\bar{a} \wedge \mathsf{F}a)$ gives the bottom-left automaton of Figure 4. Note that on this example, using the approach based on Minato's algorithm described in Section 5.3 will only give us bounds $\bar{a}...\top$ on the first transition of $A_{\mathsf{XF}a}$, but since this transition is already labeled by $\top$ it would not be changed.

As seen in this example, we propose two strategies to turn a stutter-sensitive automaton into a stutter-insensitive one. For each partly-covered equivalence class $[w]$, the relaxing strategy consists in adding the rest of $[w]$ to $\mathscr{L}(A)$. Dually, the restricting strategy consists in removing $[w]$ from $\mathscr{L}(A)$. This is legitimated by Theorem 1 provided that the added or removed parts are outside the knowledge.

To realize these strategies on automata, let us equip ourselves with a function $si(A)$ that returns an automaton A' such that $\mathscr{L}(A')$ is the smallest stutter-insensitive language that contains $\mathscr{L}(A)$. Such an operation has already been defined for Büchi automata [23] or TGBA [29]. Intuitively, it consists in two simple syntactic transformations: adding shortcut edges to reduce stutter, and adding states to allow stuttering after traversing any transition. The effect of this operation is that all classes $[w]$ partly covered by $\mathscr{L}(A)$ get fully included into $\mathscr{L}(si(A))$.

Additionally, let us define $ss(A)$ the stutter-sensitive part of A as the automaton that recognizes only the words $w \in \mathscr{L}(A)$ such that $[w] \cap \overline{\mathscr{L}(A)} \neq \emptyset$. While, to our knowledge, this operation does not exist in the literature, it can be defined using si, complement and product as follows:

$$ss(A) = A \otimes si(si(A) \otimes \overline{A})$$

In the above formula, $si(A) \otimes \overline{A}$ accepts exactly the words that should be added to $\mathscr{L}(A)$ to make it stutter-invariant. Therefore, $si(si(A) \otimes \overline{A})$ accepts all the words w such that $[w]$ is partly covered by A.

We now show how to realize our two strategies using these automata operations.

Theorem 4 (Stutter-Insensitive relaxation and restriction). *Let A be a stutter-sensitive TGBA and K be an LTL formula. We define the SI-relaxation and SI-restriction of A given K as follows.*

$$sirelax_{|K}(A) = \begin{cases} si(A) & \textit{if } \mathscr{L}(si(A) \otimes \overline{A} \otimes A_K) = \emptyset \\ A & \textit{else} \end{cases}$$

$$sirestrict_{|K}(A) = \begin{cases} A \otimes \overline{ss(A)} & \textit{if } \mathscr{L}(ss(A) \otimes A_K) = \emptyset \\ A & \textit{else} \end{cases}$$

Then $\mathscr{L}(sirelax_{|K}(A) \otimes A_K) = \mathscr{L}(sirestrict_{|K}(A) \otimes A_K) = \mathscr{L}(A \otimes A_K)$.

Proof. The proof follows from Theorem 1 and the fact that $\mathscr{L}(ss(A)) \subseteq \mathscr{L}(A) \subseteq \mathscr{L}(si(A))$. Indeed, the above relaxation returns $si(A)$ if and only if the words added by si (i.e. $\mathscr{L}(si(A) \otimes \overline{A})$) are outside $\mathscr{L}(K)$ (i.e., $si(A) \otimes \overline{A} \otimes A_K$ has an empty language). Similarly, the restriction removes from $\mathscr{L}(A)$ the words of $\mathscr{L}(ss(A))$ if and only if they are all outside of $\mathscr{L}(K)$. When the given knowledge does not allow adding or removing those words, the original automaton is returned.[5] □

Figure 4 shows stutter-insensitive automata obtained with these two constructions.

In practice the complement of A (present in both strategies) can be avoided when an LTL formula for A is known. However, the complement of $ss(A)$, needed only for $sirestrict_{|K}(A)$ can be rather costly, especially considering the current definition of $ss(A)$, which tends to create large automata. Fortunately, the latter complementation need only be performed after it has been checked that the removed words are not part of the knowledge. Our hope is therefore that any stutter-insensitive optimization performed by the model checker will offset the costs incurred by the computation of $sirestrict_{|K}(A)$.

7 Incremental Integration of Knowledge

In this section we show how to integrate knowledge when we know multiple facts about the system. We also discuss some strategies to obtain cheap knowledge tailored to help simplify a given property.

7.1 Working with a Knowledge Base

Previously, in Section 3–6, we discussed how to simplify $A_{\neg\varphi}$ given a single knowledge K. We now assume that we have multiple knowledge facts $A_{K_1}, A_{K_2}, \ldots, A_{K_n}$ about the system S and discuss strategies to integrate them all.

We could simplify $A_{\neg\varphi}$ by applying Theorems 3–4 using $A_K = A_{K_1} \otimes A_{K_2} \otimes \cdots \otimes A_{K_n}$. However, since the product of automata is quadratic in size, this automaton A_K might be very big. Even the translation of the conjunction of all K_i at once $A_K = A_{K_1 \wedge K_2 \wedge \ldots \wedge K_n}$ might be a large automaton.

In the following, we propose techniques to integrate knowledge incrementally, even if this comes with a loss of precision.

For the Boolean Bounds, we suggest applying Theorem 3 using one A_{K_i} at a time, in a loop, and delay the choice of the label (using Minato's algorithm) until the end of the loop. In the syntax of Figure 3 we do:

```
for k in list_of_facts:
   a = spot.update_bounds_given(a, k)
a_minato = spot.bounds_simplify(a)
```

Here the operation `spot.update_bounds_given(a, k)` is using the lower bounds of each transition of automaton `a` when building the product in the definition of TG and SG. In this loop, each call to `spot.update_bounds_given` may only restrict the lower bounds and relax the upper bounds of the transitions of `a`.

The incremental construction of $sirestrict_{|K}(A)$ and $sirelax_{|K}(A)$ is handled differently. Since the automata $si(A)$ and $A \otimes \overline{ss(A)}$ constructed by these techniques are

[5] The user of these functions may therefore assume that the returned automaton is stutter-insensitive whenever it is different from the input.

independent of the knowledge that allow to adopt them, we can stop as soon as we find a suitable knowledge. More formally:

$$sirelax_{|K_1,K_2,\ldots,K_n}(A) = \begin{cases} si(A) & \text{if } \exists i, \mathscr{L}(si(A) \otimes \overline{A} \otimes A_{K_i}) = \emptyset \\ A & \text{else} \end{cases}$$

$$sirestrict_{|K_1,K_2,\ldots,K_n}(A) = \begin{cases} A \otimes \overline{ss(A)} & \text{if } \exists i, \mathscr{L}(ss(A) \otimes A_{K_i}) = \emptyset \\ A & \text{else} \end{cases}$$

This generalization, which is what we implement, explains why the single fact $\underline{\mathtt{k}}$ was being passed in an array in Figure 4. In the implementation the terms $si(A) \otimes \overline{A}$ and $ss(A)$ are of course computed only once, and not for each K_i.

7.2 Seeking Knowledge

The strength of our approach is that it is agnostic to the source or proof method of the knowledge. Of course some facts might simply be other formulas we have already proven, when we are dealing with a set of specification formulas to check against a given system.

We now suggest ways to obtain some cheap knowledge about the system S, tailored to fit the formula φ that we intend to verify. For instance, we can find some simple facts on S using bounded explorations (breadth-first search, bounded model checking [3], ...), structural analysis of the system, or a decision procedure for reachability... and that can help simplify $A_{\neg\varphi}$.

Our implementation currently looks for various kinds of knowledge, using simpler decision procedures than full LTL to prove them.

Initial state First, we can check the label of the initial state of the system, giving us a knowledge of the form $\ell \in 2^{AP}$. While this knowledge is very basic, it is free. Using the initial state of the model to simplify a property has already been proposed for CTL [7].

First steps Similarly, exploring the first steps of the system is cheap. We compute the set of formulas labeling the transitions reachable in the first n steps of $A_{\neg\varphi}$, and check the first n steps of S to check if their values allow us to define a knowledge of the form $\mathsf{X}f$, $\mathsf{XX}f$...

We limit our exploration to $n = 2$ in our experiments. We use a breadth-first search with some limits to avoid explosion on models with very large branching factors ($\geq 10^4$). We could also have used any technique based on bounded model checking relying on a SAT or SMT solver.

Invariants Proving some invariants of a system can be delegated to tools that are specialized in reachability analysis and are more effective at this task than LTL model checkers.

We start by looking at the value of each atomic proposition of φ in the initial state of S and try to prove that this value never evolves using a reachability solver. We then try to evaluate compatibility of the atomic propositions, checking given two atomic propositions a and b whether all of $\bar{a}\bar{b}$, $\bar{a}b$, $a\bar{b}$ and ab are possible. For instance, $a = [x > 2]$ and $b = [x > 3]$ have a strong relationship. Knowledge about the exclusions between APs was also used by Blahoudek et al. [5]. We use an SMT solver to check if some of these cases are impossible, not even looking at the system but simply at the atomic proposition definitions. We finally also check if formulas labeling the transitions of $A_{\neg\varphi}$ are invariants. All of these strategies output knowledge of the form $\mathsf{G}f$.

Convergent atomic propositions In this approach we try to prove that a given atomic proposition a will eventually converge, providing a knowledge of the form $\mathsf{F}(\mathsf{G}a \vee \mathsf{G}\bar{a})$.

We use a low complexity structural test based on an analysis of recurring behaviors (SCC in the state graph of the system). Any atomic proposition that only observes variables in the prefix of such SCC must converge; they cannot oscillate indefinitely. We can also, using an SMT solver, try to determine the polarity of atomic propositions at convergence, yielding knowledge of the form $\mathsf{FG}a$ (or $\mathsf{FG}\bar{a}$).

These strategies are all very basic currently, but show how we can leverage a diversity of decision procedures (with lower complexity than full LTL model-checking) to populate a knowledge base that is tailored for a given formula to assist an LTL model checking step.

8 Experimental Study

Knowledge-based simplifications ("given that") have been implemented in Spot 2.13 [14]. The knowledge collection described in Section 7.2 has been implemented in ITS-tools [43], which won the LTL category of the Model Checking Contest in 2023 for the first time, thanks in part to these strategies. The tools to gather the knowledge and integrate it are open source and publicly available. A reproducibility package for the experiments can be found at `https://codeocean.com/capsule/1210152/tree/v1`.

During the competition, ITS-tools allots a small time slice to incrementally collect and integrate knowledge. After this time, it runs a portfolio of model-checkers, including a symbolic solution [43] and LTSmin [24] configured as an explicit model checker with partial-order reductions.

Measurement of the entire model checking procedure would introduce many biases due to the complex interactions of the portfolio techniques with the main refinement loop of ITS-Tools [44]. Therefore, we focus our evaluation on the knowledge integration step of the procedure, and compare the automata obtained using the strategies introduced in this paper.

8.1 Experimental Setup

The following performance analysis is based on the models and formulas of MCC'22 [25]. The benchmark uses a total of 150 different model families (coming from various domains) configured to build 1617 model instances (some models are scalable).

For each of these (colored) Petri net models, the benchmark contains 32 randomly generated LTL formulas providing a total of 51744 LTL formulas.

For each model instance, we first collected some knowledge using the basic approach of section 7.2 using ITS-Tools [43], setting a generous timeout of 15 minutes to collect it. Obtaining knowledge is cheap (median 0.67 minutes and 75% of cases below 4 minutes) as it leverages low complexity structural and symbolic tests, and in the worst case reachability queries which are much simpler than full LTL. High time usage to collect this basic knowledge correlates with models where LTL model-checking (at least in the empty product case, with no counter-example) is typically prohibitively expensive (huge models with millions of elements), so that the effort is worth it.

After this processing, we obtain some knowledge for 1601 model instances (out of 1617). For each model instance the knowledge is represented as a set of LTL assertions, for a total of 240345 small facts (roughly 150 facts per model instance). From the original set of 51744 LTL formulas we only retain 48975 formulas that intersect the gathered facts.

To add some diversity, we consider the above 48975 formulas and their negations for a total of 97950 formulas. Note that verifying an LTL formula and its negation are two independent chalenges: it can be the case that neither of these formula is verified.

For each formula, we retain only the subset of available facts whose alphabet intersects that of the formula in the experiments.[6]

Our benchmark therefore contains 97950 problems that consist in one specification LTL formula accompanied by a set of knowledge facts (on average 12.7, median 9 facts per formula).

We then proceed to apply each of our strategies to these problems to build an automaton and compute various metrics on its size. The strategies we compare are the following. A "p." used as prefix indicates a *precise* construction that considers all facts $K = \bigwedge_i K_i$ at once. While *precise* variants can pay a significant cost to manipulate the conjunction of known facts, they also benefit from a more precise knowledge so that there is a trade-off between precise and incremental approaches.

raw is formula $\neg\varphi$ translated to a TGBA, without any integration of knowledge

p.min, p.max are obtained by building $A_{\neg\varphi\wedge K}$ and $A_{\neg\varphi\vee\neg K}$ as discussed in Section 4, where K is the conjunction K of all facts.

p.min∃, p.max∃ are the variants with existential quantification shown in equations (3)–(4) from Section 4.

p.BM, BM use respectively the strategy $BM_{|K}(A)$ presented in Section 5.3, and the incremental strategy described in Section 7.1.

p.SIrelax, p.SIrestrict, SIrelax, SIrestrict are the strategies of Section 6, and their incremental variants from Section 7.1

Finally, we also consider some combination of techniques. For instance "SIrelax+BM" designates the incremental implementation of SIrelax followed by the incremental implementation of BM.

[6] This is not necessarily optimal. Consider $\varphi = \mathsf{GF}a$ with alphabet $\{a\}$ and facts $k_1 = \mathsf{G}(b \rightarrow \mathsf{X}a)$ and $k_2 = \mathsf{FG}b$, ignoring k_2 because its alphabet does not intersect φ's is in fact a mistake ; however selecting too many facts can easily overload some approaches particularly those using the conjunction of known facts, and does not help our incremental approaches that consider each fact in isolation since they typically ignore atomic propositions not in φ.

Table 1: Amount of problems (out of 97950 composed of formulas and their negation) that could be shown to be empty or universal using the provided knowledge.

Strategy	Universal	Empty	Total
p.min	0	25508	25508
p.max	24453	0	24453
p.min∃	0	25508	25508
p.max∃	25508	0	25508
BM	23080	25095	48175
SIrelax	208	0	208
SIrestrict	0	219	219
p.BM	23258	25508	48766
p.SIrelax	212	0	212
p.SIrestrict	0	223	223
SIrelax+BM	23286	25091	48377
BM+SIrelax	23164	25095	48259
p.SIrelax+p.BM	23708	25508	49216
p.BM+p.SIrelax	23344	25508	48852

When providing statistics about the automata produced by the above variants, we always assume that those automata have been further simplified using techniques implemented in Spot (notably, removing useless states, useless acceptance marks, and using simulation-based reductions to merge states and prune unnecessary transitions [2]).

The average runtime for solving a problem with any strategy is 35.6ms. On the 97950 benchmark problems, the only strategies that exceed a very generous timeout of 10 seconds are “SIrestrict” on 470 problems, and “p.SIrestrict” on 522 problems. Those strategies are occasionally very slow only because of the amount of automata complementations they have to perform. Overall the knowledge integration step is truly negligible before any test involving the actual system. If the knowledge gathering step only uses low complexity procedures or knowledge simply consists of previously proven properties, knowledge integration scales exceptionally well to complex problems.

8.2 Problems Reduced to Empty or Universal

We first study the problems that could be fully solved given the knowledge by reducing the automaton to an empty or universal one. For testing universality, we syntactically check if the resulting automaton has been reduced to a single-state all-accepting automaton.

Table 1 presents those results. While it is certainly due to the random nature of the formulas of the MCC, in total $51016/97950 \approx 52\%$ of the formulas of the MCC benchmark we kept (49% of all formulas of the MCC) could be solved using only the basic approach to glean related knowledge presented in Section 7.2, thus avoiding a full LTL model-checking procedure.

We can see that “min∃” and “p.min∃” are the most effective strategies to find empty problems, on par with “p.BM”. Dually “p.max∃” is the only most effective at deducing universal problems. The amount of problems reduced to empty by “p.min∃” and to universal by “p.max∃” are identical as hoped, because our benchmark includes both formulas and their negations. Strategy “p.max” is less effective than “p.max∃” because it keeps atomic propositions that are not in φ. Still, while they might produce a larger automaton as discussed in the next section if they can’t solve the problem, it is important in a full decision approach involving some knowledge to first test “p.min∃” and “p.max∃” for full solutions.

Table 2: Comparison of the different strategies over 46934 problems that could not be already reduced to false or true by previous methods. ‘raw’ designates the original automata, for baseline. For each strategy we report various metrics of the produced automata: its number of states and transitions, $\sum |f|$ is the total size of all labels, SI (resp. det) shows the fraction of automata that were stutter-insensitive (resp. deterministic), $|AP|$ is the number of atomic proposition, Time reports the number of milliseconds needed by the strategy, and TO counts the number of timeouts (> 10 seconds). Different statistics are provided for some measurements: ‘q95’ denotes the 95% quantile (i.e., 95% of all values are below the indicated value), ‘geom’ denotes the geometric mean. Values within 2% of the best (resp. worse) value of a column, ‘raw‘ excluded, are highlighted in yellow (resp. pink).

	States				Transitions				$\lvert AP\rvert$	$\sum \lvert f\rvert$	SI	det	Time (ms)		
Strategy	q95	max	mean	geom	q95	max	mean	geom	mean	mean			mean	geom	TO
raw	8	73	3.71	3.13	18	286	7.32	5.52	2.14	21.76	49%	50%	20.26	19.95	0
p.min	9	684	5.30	4.79	19	16834	8.71	6.71	3.35	48.36	9%	50%	55.04	49.82	0
p.max	10	76	6.30	5.90	25	295	12.78	11.23	3.35	58.15	9%	43%	55.86	50.27	0
p.min∃	9	684	5.23	4.67	19	16834	8.60	6.55	2.21	35.06	11%	50%	54.18	48.97	0
p.max∃	10	76	6.14	5.73	25	295	12.37	10.77	2.21	42.54	11%	45%	54.91	49.39	0
BM	6	65	3.13	2.68	13	286	5.42	4.24	1.70	13.39	46%	59%	41.46	40.62	0
SIrelax	8	73	3.86	3.16	22	286	8.07	5.82	2.14	26.51	66%	49%	42.19	41.09	0
SIrestrict	10	19463	6.77	3.23	27	373093	52.25	5.96	2.14	335.58	67%	51%	57.65	44.28	122
p.BM	6	65	3.13	2.68	13	286	5.41	4.24	1.69	13.35	46%	59%	46.13	43.50	0
p.SIrelax	9	73	3.93	3.18	24	340	8.47	5.92	2.14	29.15	70%	49%	46.20	43.71	0
p.SIrestrict	11	84249	10.59	3.28	32	1252969	105.47	6.12	2.14	677.23	70%	51%	57.34	45.53	130
SIrelax+BM	6	65	3.07	2.62	13	286	5.38	4.18	1.70	13.49	51%	59%	63.37	61.75	0
BM+SIrelax	7	65	3.19	2.67	15	286	5.93	4.49	1.70	16.21	67%	58%	63.15	61.46	0
p.SIrelax+p.BM	6	65	3.04	2.59	13	286	5.34	4.15	1.69	13.44	52%	59%	72.08	66.89	0
p.BM+p.SIrelax	7	65	3.16	2.63	16	286	5.99	4.46	1.69	16.83	70%	58%	71.85	66.69	0

Generally, strategies based only on restriction (resp. relaxation) can prove only emptiness (resp. universality) and obtaining “empty” seems easier on this benchmark than obtaining “universal” perhaps due to our limited syntactic check for universality.

8.3 Simplifying the Remaining Unsolved Formulas

We now study in Table 2 statistics for all the 46934 problems that could not be proven empty or universal by any strategy.

Since our goal is to reduce the size of the automaton, we first study the number of states and transitions. To better understand the distribution of values, we present the 95% quantile, as well as the arithmetic and geometric means. Cases where the arithmetic mean is much larger than the geometric mean indicate the presence of a few very large outliers.

We observe that basic strategies based on “p.min” or “p.max” are not very good, as feared, doubling the average size. However, all strategies involving “BM” perform well: the average number of state is reduced by 15%, and transitions by 25%. The “SIrelax” strategy produces a moderate size increase that can be further alleviated by combining it with “BM”. However, “SIrestrict” can dramatically increase the size of the automaton, and even time out in extreme cases.

The number of atomic propositions (column $|AP|$) and size of the formula labels ($\sum |f|$) is also significantly reduced by all variants using “BM”. The average number of atomic propositions is reduced from 2.14 to around 1.7 (a 21% gain), and the average size of formula labels goes from 21.76 to around 13.4 (a 38% gain).

Concerning the stutter insensitivity (column “SI”) of the resulting automaton, only 49% of the “raw” problems are stutter-insensitive. “min” and “max” degrade this number significantly. Both of the strategies “SIrestrict” and “SIrelax” developed to optimize this metric are indeed effective (but “SIrestrict” is more expensive and liable to timeout). Combined strategies that finish with a “SIrelax” step lead to the best results, being both small and stutter-insensitive in 70% of cases. The precise variants are a bit better than the incremental constructions.

While our algorithm is not looking to improve determinism (column “det”), this characteristic is nonetheless improved by all variants involving “BM”. This is a welcome side-effect since having small *and* deterministic automata can only help model checking [39,4].

On this subset of 46934 cases, the average time to solve a problem is up to two times higher than the average of the 97950 benchmark problems (which was 35.6ms as mentioned earlier). However it is still very cheap. The fastest strategies to integrate knowledge is “BM” (with an average of 41ms). “p.” precise variants all pay a reasonable time penalty, but the improvement in size is very modest. The only timeouts we observe on these problems that cannot be entirely solved are for “SIrestrict” and its precise variant (in less than 3‰ cases).

In conclusion, combined strategies using “SIrelax” and “BM” produce the smallest automata without real drawbacks apart from moderate increase of the run time. Given the very reasonable run times, it is even feasible to run several of these strategies and then select the most appropriate automaton on a case by case basis.

9 Related Work

Theorem 1 proposes an original framework for exploiting prior knowledge in LTL model checking. This generalizes approaches that only consider invariants [7] or quasi-invariants [27].

Theorem 1 is also related to the problem of language separation: given two languages, a separator is a third language that contains the first one and is disjoint from the second one [34]. In our case, we are looking for an automaton B whose language separates $\mathscr{L}(A) \cap \mathscr{L}(K)$ (which it should include) from $\mathscr{L}(K) \setminus \mathscr{L}(A)$ (which it should not intersect). However, the two languages to separate aren’t independent: A is already known to be a separator, and we are trying to find a simpler B by simplifying A.

Blaoudek et al. [5, Section 5] also consider a simplification of labels leveraging Minato’s algorithm as we did in Section 5. While it is limited to an invariant about mutually exclusive propositions, it did prove to be an effective simplification. Our approach generalizes theirs: if the knowledge encodes mutual exclusion of atomic propositions, we will generate the same bounds, however we can handle arbitrary LTL knowledge, and we take the structure of the automaton into account.

Using Minato's algorithm to find a simple f' such that $f_{low} \Rightarrow f' \Rightarrow f_{high}$ can be related to Coudert and Madre's `restrict` and `constraint` operators [10] that find f' such that $f \wedge c \Rightarrow f' \Rightarrow f \vee \neg c$, where c is a Boolean formula. However, in our case, f_{low} and f_{high} are not limited to this form.

The use of bounded automata in Section 5 evokes the notion of incompletely specified Mealy machines used in synthesis, where "don't care" edges are leveraged to produce smaller automata [31,1,36]. The bounded automata we propose can be used for bound-aware simulation-based reductions [41]; this could complement our current approaches.

Dureja and Rozier [12] consider the problem of model checking a single model against a large set of LTL formulas. They compute a matrix of implications between formulas $f_i \Rightarrow f_j$, and they use previously proven formula f_1 to avoid model checking of implied formulas f_2. Such an implication test, between a previously proven formula f_1 (the knowledge) and an unproved formula f_2, is covered in our approach since "f_2 given f_1" will be an empty automaton (see Section 8.2). However, we can also obtain a simpler automaton for f_2 even in the absence of full implication. Moreover, we suggest several approaches to leverage *all* accumulated knowledge incrementally.

Our definitions suggest explicit representation of automata, however our approach can be used for symbolic model checking. Instead of using a direct symbolic encoding of Büchi automaton [37], obtained directly from LTL, we can encode the explicit automaton resulting from our knowledge simplifications into a symbolic representation [40]. In fact, ITS-tools uses both a knowledge-based approach and a symbolic encoding.

Although this work is motivated by model checking, our techniques can be used to optimize any inclusion check $\mathscr{L}(A) \subseteq \mathscr{L}(B)$. E.g., in the traditional implementation based on a complementation [45] of B, any knowledge about A, can be used to simplify B before its complementation.

10 Conclusion

We have introduced new operations that help simplify the model-checking of a new formula when we already possess some prior knowledge on the system. Our strategies are automata-based operations, thus capturing any nature of LTL property or prior knowledge. The evaluation of our current implementation on a large benchmark demonstrates the effectiveness of the approach.

Studying the problem of knowledge integration led us to the problem of producing a (small) automaton given bounds on the language it represents. This challenging problem is new to our knowledge and while we have proposed several strategies in this paper, there is a lot of room for more research in this direction. For instance the strategies we presented in Section 6 to produce stutter-insensitive automata currently do not take any advantage of the Boolean bounds computed in Section 5. Similarly, those Boolean bounds could very likely be used for other kinds of simplifications, such as bound-aware simulation-based reductions [41]. The problem of seeking relevant knowledge by leveraging simpler decision procedures than full LTL is also an avenue for further exploration, paving the way to strategies achieving an incremental verification process.

References

1. Abel, A., Reineke, J.: MeMin: SAT-based exact minimization of incompletely specified Mealy machines. In: Proceedings for the 34th International Conference on Computer-Aided Design (ICCAD'15). pp. 94–101. IEEE Press (2015). `https://doi.org/10.1109/ICCAD.2015.7372555`
2. Babiak, T., Badie, T., Duret-Lutz, A., Křetínský, M., Strejček, J.: Compositional approach to suspension and other improvements to LTL translation. In: Proceedings of the 20th International SPIN Symposium on Model Checking of Software (SPIN'13). Lecture Notes in Computer Science, vol. 7976, pp. 81–98. Springer (Jul 2013). `https://doi.org/10.1007/978-3-642-39176-7_6`
3. Biere, A.: Bounded model checking. In: Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.) Handbook of Satisfiability - Second Edition, Frontiers in Artificial Intelligence and Applications, vol. 336, pp. 739–764. IOS Press (2021). `https://doi.org/10.3233/FAIA201002`
4. Blahoudek, F., Duret-Lutz, A., Křetínský, M., Strejček, J.: Is there a best Büchi automaton for explicit model checking? In: Proceedings of the 21th International SPIN Symposium on Model Checking of Software (SPIN'14). pp. 68–76. ACM (Jul 2014). `https://doi.org/10.1145/2632362.2632377`
5. Blahoudek, F., Duret-Lutz, A., Rujbr, V., Strejček, J.: On refinement of Büchi automata for explicit model checking. In: Proceedings of the 22th International SPIN Symposium on Model Checking of Software (SPIN'15). Lecture Notes in Computer Science, vol. 9232, pp. 66–83. Springer (Aug 2015). `https://doi.org/10.1007/978-3-319-23404-5_6`
6. Bloem, R., Ravi, K., Somenzi, F.: Efficient decision procedures for model checking of linear time logic properties. In: Proceedings of the Eleventh Conference on Computer Aided Verification (CAV'99). Lecture Notes in Computer Science, vol. 1633, pp. 222–235. Springer-Verlag (1999)
7. Bønneland, F., Dyhr, J., Jensen, P.G., Johannsen, M., Srba, J.: Simplification of CTL formulae for efficient model checking of Petri nets. In: Petri Nets. LNCS, vol. 10877, pp. 143–163. Springer (2018)
8. Bryant, R.E.: Graph-based algorithms for boolean function manipulation. IEEE Transactions on Computers **35**(8), 677–691 (Aug 1986)
9. Černá, I., Pelánek, R.: Relating hierarchy of temporal properties to model checking. In: Rovan, B., Vojtáă, P. (eds.) Proceedings of the 28th International Symposium on Mathematical Foundations of Computer Science (MFCS'03). Lecture Notes in Computer Science, vol. 2747, pp. 318–327. Springer-Verlag, Bratislava, Slovak Republic (Aug 2003)
10. Coudert, O., Madre, J.C.: A unified framework for the formal verification of sequential circuits. In: Proceedings of the International Conference on Computer-Aided Design (ICCAD'90). pp. 126–129 (1990). `https://doi.org/10.1109/ICCAD.1990.129859`
11. Couvreur, J.M.: On-the-fly verification of temporal logic. In: Wing, J.M., Woodcock, J., Davies, J. (eds.) Proceedings of the World Congress on Formal Methods in the Development of Computing Systems (FM'99). Lecture Notes in Computer Science, vol. 1708, pp. 253–271. Springer-Verlag, Toulouse, France (Sep 1999)
12. Dureja, R., , Rozier, K.Y.: More scalable LTL model checking via discovering design-space dependencies (D^3). In: Proceedings of the 24th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'24). pp. 309–327. Springer International Publishing, Cham (2018). `https://doi.org/10.1007/978-3-319-89960-2_17`
13. Duret-Lutz, A.: LTL translation improvements in Spot 1.0. International Journal on Critical Computer-Based Systems **5**(1/2), 31–54 (Mar 2014). `https://doi.org/10.1504/IJCCBS.2014.059594`

14. Duret-Lutz, A., Renault, E., Colange, M., Renkin, F., Aisse, A.G., Schlehuber-Caissier, P., Medioni, T., Martin, A., Dubois, J., Gillard, C., Lauko, H.: From Spot 2.0 to Spot 2.10: What's new? In: Proceedings of the 34th International Conference on Computer Aided Verification (CAV'22). Lecture Notes in Computer Science, vol. 13372, pp. 174–187. Springer (Aug 2022). https://doi.org/10.1007/978-3-031-13188-2_9
15. Etessami, K., Holzmann, G.J.: Optimizing Büchi automata. In: Palamidessi, C. (ed.) Proceedings of the 11th International Conference on Concurrency Theory (Concur'00). Lecture Notes in Computer Science, vol. 1877, pp. 153–167. Springer-Verlag, Pennsylvania, USA (2000)
16. Etessami, K., Wilke, T., Schuller, R.A.: Fair simulation relations, parity games, and state space reduction for Büchi automata. In: Orejas, F., Spirakis, P.G., van Leeuwen, J. (eds.) Proceedings of the 28th international colloquium on Automata, Languages and Programming. Lecture Notes in Computer Science, vol. 2076, pp. 694–707. Springer-Verlag, Crete, Greece (Jul 2001)
17. Fogarty, S., Vardi, M.Y.: Efficient Büchi universality checking. In: Proceedings of the 16th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'10). pp. 205–220. Springer Berlin Heidelberg, Berlin, Heidelberg (2010). https://doi.org/10.1007/978-3-642-12002-2_17
18. Fritz, C.: Constructing Büchi automata from linear temporal logic using simulation relations for alternating Büchi automata. In: Ibarra, O.H., Dang, Z. (eds.) Proceedings of the 8th International Conference on Implementation and Application of Automata (CIAA'03). Lecture Notes in Computer Science, vol. 2759, pp. 35–48. Springer-Verlag, Santa Barbara, California (Jul 2003)
19. Gastin, P., Oddoux, D.: Fast LTL to Büchi automata translation. In: Berry, G., Comon, H., Finkel, A. (eds.) Proceedings of the 13th International Conference on Computer Aided Verification (CAV'01). Lecture Notes in Computer Science, vol. 2102, pp. 53–65. Springer-Verlag, Paris, France (2001). https://doi.org/10.1007/3-540-44585-4_6
20. Giannakopoulou, D., Lerda, F.: From states to transitions: Improving translation of LTL formulæ to Büchi automata. In: Peled, D., Vardi, M. (eds.) Proceedings of the 22nd IFIP WG 6.1 International Conference on Formal Techniques for Networked and Distributed Systems (FORTE'02). Lecture Notes in Computer Science, vol. 2529, pp. 308–326. Springer-Verlag, Houston, Texas (Nov 2002)
21. Godefroid, P.: Partial-Order Methods for the Verification of Concurrent Systems: An Approach to the State-Explosion Problem, Lecture Notes in Computer Science, vol. 1032. Springer-Verlag (1996)
22. Haddad, S., Pradat-Peyre, J.F.: New efficient Petri nets reductions for parallel programs verification. Parallel Processing Letters **16**(01), 101–116 (2006)
23. Holzmann, G.J., Kupferman, O.: Not checking for closure under stuttering. In: Proceedings of the 2nd workshop on the Spin Verification System (SPIN'96). pp. 17–22. American Mathematical Society (1996)
24. Kant, G., Laarman, A., Meijer, J., van de Pol, J., Blom, S., van Dijk, T.: Ltsmin: High-performance language-independent model checking. In: Baier, C., Tinelli, C. (eds.) Proceedings of the 21st conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'15). pp. 692–707. Springer Berlin Heidelberg (2015). https://doi.org/10.1007/978-3-662-46681-0_61
25. Kordon, F., Bouvier, P., Garavel, H., Hulin-Hubard, F., Amat., N., Amparore, E., Berthomieu, B., Donatelli, D., Dal Zilio, S., Jensen, P., Jezequel, L., He, C., Li, S., Paviot-Adet, E., Srba, J., Thierry-Mieg, Y.: Complete Results for the 2022 Edition of the Model Checking Contest. http://mcc.lip6.fr/2022/results.php (June 2022)

26. Kordon, F., Hillah, L.M., Hulin-Hubard, F., Jezequel, L., Paviot-Adet, E.: Study of the efficiency of model checking techniques using results of the model-checking contest mcc from 2015 to 2019. International Journal on Software Tools for Technology Transfer (2021). `https://doi.org/10.1007/s10009-021-00615-1`, `https://doi.org/10.1007/s10009-021-00615-1`
27. Larraz, D., Nimkar, K., Oliveras, A., Rodríguez-Carbonell, E., Rubio, A.: Proving non-termination using Max-SMT. In: Biere, A., Bloem, R. (eds.) Proceeding of the 26th International Conference on Computer Aided Verification (CAV'14). pp. 779–796. Springer International Publishing, Cham (2014). `https://doi.org/10.1007/978-3-319-08867-9_52`
28. Manna, Z., Pnueli, A.: A hierarchy of temporal properties. In: Proceedings of the sixth annual ACM Symposium on Principles of distributed computing (PODC'90). pp. 377–410. ACM, New York, NY, USA (1990)
29. Michaud, T., Duret-Lutz, A.: Practical stutter-invariance checks for ω-regular languages. In: Proceedings of the 22th International SPIN Symposium on Model Checking of Software (SPIN'15). Lecture Notes in Computer Science, vol. 9232, pp. 84–101. Springer (Aug 2015). `https://doi.org/10.1007/978-3-319-23404-5_7`
30. Minato, S.: Fast generation of irredundant sum-of-products forms from binary decision diagrams. In: Proceedings of the third Synthesis and Simulation and Meeting International Interchange workshop (SASIMI'92). pp. 64–73. Kobe, Japan (Apr 1992)
31. Paull, M.C., Unger, S.H.: Minimizing the number of states in incompletely specified sequential switching functions. IRE Transactions on Electronic Computers **EC-8**(3), 356–367 (Sep 1959). `https://doi.org/10.1109/TEC.1959.5222697`
32. Peled, D.: Combining partial order reductions with on-the-fly model-checking. In: Proceedings of the 6th International Conference on Computer Aided Verification (CAV'94). Lecture Notes in Computer Science, vol. 818, pp. 377–390. Springer-Verlag (1994)
33. Peled, D., Wilke, T., Wolper, P.: An algorithmic approach for checking closure properties of temporal logic specifications and ω-regular languages. Theoretical Computier Science **195**(2), 183–203 (Mar 1998). `https://doi.org/10.1016/S0304-3975(97)00219-3`
34. Place, T., Zeitoun, M.: Separating regular languages with first-order logic. Logical Methods in Computer Science **Volume 12, Issue 1** (Mar 2016). `https://doi.org/10.2168/lmcs-12(1:5)2016`
35. Renault, E., Duret-Lutz, A., Kordon, F., Poitrenaud, D.: Strength-based decomposition of the property Büchi automaton for faster model checking. In: Piterman, N., Smolka, S.A. (eds.) Proceedings of the 19th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'13). Lecture Notes in Computer Science, vol. 7795, pp. 580–593. Springer (Mar 2013). `https://doi.org/10.1007/978-3-642-36742-7_42`
36. Renkin, F., Schlehuber-Caissier, P., Duret-Lutz, A., Pommellet, A.: Effective reductions of Mealy machines. In: Proceedings of the 42nd International Conference on Formal Techniques for Distributed Objects, Components, and Systems (FORTE'22). Lecture Notes in Computer Science, Springer (Jun 2022), to appear
37. Rozier, K.Y., Vardi, M.Y.: A multi-encoding approach for LTL symbolic satisfiability checking. In: Butler, M.J., Schulte, W. (eds.) Proceedings of the 17th International Symposium on Formal Methods (FM'11). Lecture Notes in Computer Science, vol. 6664, pp. 417–431. Springer (2011). `https://doi.org/10.1007/978-3-642-21437-0_31`
38. Schewe, S., Varghese, T.: Tight bounds for the determinisation and complementation of generalised Büchi automata. In: Chakraborty, S., Mukund, M. (eds.) Proceedings of the 10th International Symposium on Automated Technology for Verification and Analysis (ATVA'12). Lecture Notes in Computer Science, vol. 7561, pp. 42–56. Springer (Oct 2012). `https://doi.org/10.1007/978-3-642-33386-6_5`

39. Sebastiani, R., Tonetta, S.: "more deterministic" vs. "smaller" Büchi automata for efficient LTL model checking. In: Goos, G., Hartmanis, J., van Leeuwen, J. (eds.) Proceedings of the 12th Advanced Research Working Conference on Correct Hardware Design and Verification Methods (CHARME'03). Lecture Notes in Computer Science, vol. 2860, pp. 126–140. Springer-Verlag, L'Aquila, Italy (Oct 2003)
40. Sebastiani, R., Tonetta, S., Vardi, M.Y.: Symbolic systems, explicit properties: on hybrid approches for LTL symbolic model checking. In: Etessami, K., Rajamani, S.K. (eds.) Proceedings of 17th International Conference on Computer Aided Verification (CAV'05). Lecture Notes in Computer Science, vol. 3576, pp. 350–363. Springer, Edinburgh, Scotland, UK (Jul 2005)
41. Smolka, D.: Simulation-Based Reduction of Modal Omega-Automata. Bachelor's thesis, Mazaryk University, Faculty of Informatics (Apr 2023), `https://is.muni.cz/th/qq7ad/?lang=en`
42. Somenzi, F., Bloem, R.: Efficient Büchi automata for LTL formulæ. In: Proceedings of the 12th International Conference on Computer Aided Verification (CAV'00). Lecture Notes in Computer Science, vol. 1855, pp. 247–263. Springer-Verlag, Chicago, Illinois, USA (2000)
43. Thierry-Mieg, Y.: Symbolic model-checking using ITS-tools. In: TACAS. LNCS, vol. 9035, pp. 231–237. Springer (2015)
44. Thierry-Mieg, Y.: Structural reductions revisited. In: Petri Nets. LNCS, vol. 12152, pp. 303–323. Springer (2020)
45. Tsai, M.H., Fogarty, S., Vardi, M.Y., Tsay, Y.K.: State of Büchi complementation. Logical Methods in Computer Science **Volume 10, Issue 4** (Dec 2014). `https://doi.org/10.2168/lmcs-10(4:13)2014`
46. Valmari, A.: On-the-fly verification with stubborn sets. In: Proceedings of the 5th International Conference on Computer Aided Verification (CAV '93). pp. 397–408. Springer-Verlag, London, UK (1993)
47. Vardi, M.Y.: Automata-theoretic model checking revisited. In: VMCAI. LNCS, vol. 4349, pp. 137–150. Springer (2007)
48. Yan, Q.: Lower bounds for complementation of omega-automata via the full automata technique. Logical Methods in Computer Science **4**(1) (Mar 2008)

The following appendix is not meant to be part of the published paper because of size restrictions. It is included for the benefit of the interested reviewers.

A Proof of Theorem 3

Theorem 3. *Using the same automata as in Definition 3, consider a transition* $t = q \xrightarrow{f,a} q' \in \delta$*, and let* $B = \langle AP, Q, \iota, Acc, \delta \setminus \{t\} \cup \{t'\}\rangle$ *be a copy of* $A_{\neg\varphi}$ *where* t *has been replaced by* $t' = q \xrightarrow{f',a} q'$ *where* $f' \in \mathbb{B}(AP)$ *is any formula such that*

$$\underbrace{f \wedge \mathsf{TG}(t)}_{\textit{lower bound}} \Rightarrow f' \Rightarrow \underbrace{f \vee \neg\mathsf{SG}(q)}_{\textit{upper bound}}$$

Then $\mathscr{L}(B \otimes A_K) = \mathscr{L}(A_{\neg\varphi} \otimes A_K)$

Proof. As a preliminary, notice that since B and $A_{\neg\varphi}$ differ only by the labels of their transitions, but the product $B \otimes A_K$ and $A_{\neg\varphi} \otimes A_K$ have the same states.

($\supseteq$) Consider a word $w = \ell_0\ell_1\ell_2 \ldots \in \mathscr{L}(A_{\neg\varphi} \otimes A_K)$. There exists an accepting run $r = (q_1, q_{k1}) \xrightarrow[t_1]{f_1 \wedge f_{k1}, a_1 \cup a_{k1}} (q_2, q_{k2}) \xrightarrow[t_2]{f_2 \wedge f_{k2}, a_2 \cup a_{k2}} \cdots$ of $A_{\neg\varphi} \otimes A_K$ such that for all i, we have $\ell_i \Rightarrow f_i \wedge f_{ki}$. We named the transitions $t_1, t_2, \ldots$ for later reference. By definition of the product, this run can be seen a the synchronization of two runs: $r_A = q_1 \xrightarrow[t_{a1}]{f_1, a_1} q_2 \xrightarrow[t_{a2}]{f_2, a_2} \cdots$ a run of $A_{\neg\varphi}$ accepting w, and $r_K = q_{k1} \xrightarrow{f_{k1}, a_{k1}} (q_{k2}) \xrightarrow{f_{k2}, a_{k2}} \cdots$ a run of A_K accepting w as well.

Since B has been constructed from $A_{\neg\varphi}$ by just changing the labels of the edges to anything permitted by the theorem, B necessarily contains a run $r_B = q_1 \xrightarrow{f'_1, a_1} q_2 \xrightarrow{f'_2, a_2} \cdots$ such that $f_i \wedge \mathsf{TG}(t_{ai}) \Rightarrow f'_i$ for each i. This run r_B is accepting because it sees the same acceptance marks as r_A. We claim that r_b is an accepting run on w because it can be shown that $\ell_i \Rightarrow f_i \wedge \mathsf{TG}(t_{ai}) \Rightarrow f'_i$.

Since it is already the case that $\ell_i \Rightarrow f_i$ for each i (since r_A accepts w), we just have to prove that $\ell_i \Rightarrow \mathsf{TG}(t_{ai})$. The transition t_i (which was obtained by synchronizing t_{ai} and t_{ki}) is part of the accepting run r, so it also belongs to $\mathit{Trim}(A_{\neg\varphi} \otimes A_K)$. This means that $\mathsf{TG}(t_{ai})$ contains at least f_{ki} as a disjunct. We can therefore say that $\ell_i \Rightarrow f_{ki} \Rightarrow \mathsf{TG}(t_{ai})$ for all i. Conclusion: w is still accepted by B and therefore by $B \otimes A_K$ as well.

($\subseteq$) Consider an accepted word $w = \ell_0\ell_1\ell_2 \ldots \in \mathscr{L}(B \otimes A_K)$. There exists an accepting run $r' = (q_1, q_{k1}) \xrightarrow[t'_1]{f'_1 \wedge f_{k1}, a_1 \cup a_{k1}} (q_2, q_{k2}) \xrightarrow[t'_2]{f'_2 \wedge f_{k2}, a_2 \cup a_{k2}} \cdots$ of $B \otimes A_K$ such that for all i, we have $\ell_i \Rightarrow f'_i \wedge f_{ki}$.

By definition of the product, this run can be seen a the synchronization of two runs: $r_B = q_1 \xrightarrow[t_{b1}]{f'_1, a_1} q_2 \xrightarrow[t_{b2}]{f'_2, a_2} \cdots$ a run of B, and $r_K = q_{k1} \xrightarrow{f_{k1}, a_{k1}} q_{k2} \xrightarrow{f_{k2}, a_{k2}} \cdots$ a run of A_K, both accepting w.

Because of the way B has been constructed in the Theorem, we know that for each transition t_{bi} there is a corresponding transition $q_i \xrightarrow{f_i, a_i} q_{i+1}$ in $A_{\neg\varphi}$ such that

$f'_i \Rightarrow f_i \vee \neg\,\mathsf{SG}(q_i)$. Therefore, since $\ell_i \Rightarrow f'_i$ we have $(\ell_i \Rightarrow f_i) \vee (\ell_i \Rightarrow \neg\,\mathsf{SG}(q_i))$. Let us show that the latter clause is false, so that the former one needs to be true.

Since $\mathsf{SG}(q_i)$ is the disjunction of all labels that A_K could do when it is synchronized with q_i, let us *assume* that (q_i, q_{ki}) is reachable in $\mathit{Trim}(A_{\neg\varphi} \otimes A_K)$. Then the transition $q_{ki} \xrightarrow{f_{ki}, a_{ki}} q_{k(i+1)}$ is considered when building the disjuncts of $\mathsf{SG}(q_i)$, so we have $\ell_i \Rightarrow f_{ki} \Rightarrow \mathsf{SG}(q_i)$. This implies that $\ell_i \not\Rightarrow \neg\,\mathsf{SG}(q_i)$, which, combined with the last equation of the previous paragraph implies that $\ell_i \Rightarrow f_i$.

We conclude that if (q_i, q_{ki}) is reachable, then not only $\ell_i \Rightarrow f_i$, but also the transition $(q_i, q_{ki}) \xrightarrow{f_i \wedge f_{ki}, a_i \cup a_{ki}} (q_{i+1}, q_{k(i+1)})$ exists in $A_{\neg\varphi} \otimes A_K$ making $(q_{i+1}, q_{k(i+1)})$ reachable in turn.

Since the initial state is obviously reachable, the above reasoning allows us to inductively define a run $r = (q_1, q_{k1}) \xrightarrow{f_1 \wedge f_{k1}, a_1 \cup a_{k1}} (q_2, q_{k2}) \xrightarrow{f_2 \wedge f_{k2}, a_2 \cup a_{k2}} \cdots$ of $A_{\neg\varphi} \otimes A_K$ that accepts w. $\square$