On the Virtue of Patience: Minimizing Büchi Automata[⋆]

Rüdiger Ehlers and Bernd Finkbeiner

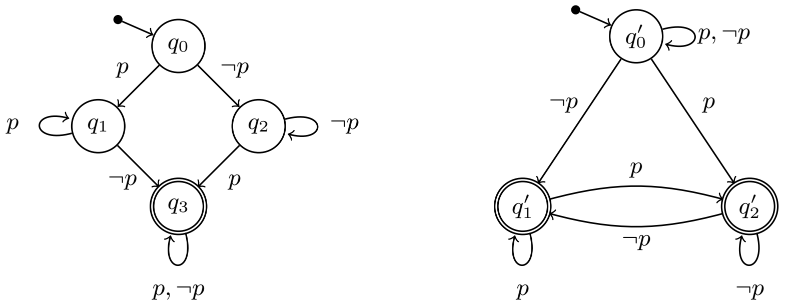
Reactive Systems Group
Saarland University
{ehlers,finkbeiner}@react.cs.uni-saarland.de

Abstract. Explicit-state model checkers like SPIN, which verify systems against properties stated in linear-time temporal logic (LTL), rely on efficient LTL-to-Büchi translators. A difficult design decision in such constructions is to trade time spent on minimizing the Büchi automaton versus time spent on model checking against an unnecessarily large automaton. Standard reduction methods like simulation quotienting are fast but often miss optimization opportunities. We propose a new technique that achieves significant further reductions when more time can be invested in the minimization of the automaton. The additional effort is often justified, for example, when the properties are known in advance, or when the same property is used in multiple model checking runs. We use a modified SAT solver to perform bounded language inclusion checks on partial solutions. SAT solving allows us to prune large parts of the search space for smaller automata already in the early solving stages. The bound allows us to fine-tune the algorithm to run in limited time. Our experimental results show that, on standard LTL-to-Büchi benchmarks, our prototype implementation achieves a significant further size reduction on automata obtained by the best currently available LTL-to-Büchi translators.

1 Introduction

Minimizing Büchi automata is a fundamental task in automatic verification. Explicit-state model checkers like SPIN [13] translate the specification, given as a formula in linear-time temporal logic (LTL), into a Büchi automaton that corresponds to the negation of the formula. This automaton is composed with the system-under-verification, and the resulting product is checked for counterexample traces. Since the specification is usually much smaller than the system, reducing the automaton generated from the specification even by a small number of states can have a huge impact on the size of the product state space and, hence, on the performance of the model checker.

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



(a) $\mathcal{A}$ (b) $\mathcal{A}'$

Fig. 1. Two Büchi automata for the LTL formula $(Fp) \wedge (F\neg p)$. The 4-state automaton $\mathcal{A}$ shown on the left was generated by a standard LTL-to-Büchi translator. The equivalent 3-state automaton $\mathcal{A}'$ shown on the right was obtained using our minimization method.

Since minimizing Büchi automata is hard (it includes the universality problem as a special case, which is already PSPACE-hard), the optimization techniques in the literature aim for a compromise between time spent on minimizing the Büchi automaton versus time spent on model checking against an unnecessarily large automaton. In addition to simple local optimizations such as edge label rewriting and the elimination of subautomata without accepting runs, the standard approach is to merge states based on simulation and bisimulation relations (cf. [8, 16, 12]). Simulation and bisimulation relations can be computed in polynomial time. Optimizing automata by merging similar states is therefore fast, but it often misses optimization opportunities. Consider, for example, the automata shown in Figure 1, which correspond to the LTL formula $\mathsf{F}p \wedge \mathsf{F}\neg p$ over the singleton set $\{p\}$ of atomic propositions. The 4-state automaton $\mathcal{A}$ on the left was generated by a standard LTL-to-Büchi translator: q_0 is the initial state; q_1 represents the case where a p has been seen, but not yet a $\neg p$, and, analogously, q_2 the case where a $\neg p$ has been seen but not yet a p; q_3 represents the case where both obligations have been satisfied. Since none of the four states simulates any other state, standard reduction techniques cannot optimize this automaton any further. There exists, however, an equivalent automaton $\mathcal{A}'$ with just 3 states, shown on the right in Figure 1: an accepting run on a word that contains both p and $\neg p$ simply stays in the initial state q'_0 until just before some $p, \neg p$ or $\neg p, p$ sequence occurs and then moves between the accepting states q'_1 and q'_2 according to the remaining suffix.

Given the complexity of the minimization problem, it seems unlikely that one can improve over the standard reduction techniques without increasing the computation time. However, investing additional time may well be justified. Considering that the specification is often written long before the system design is complete, it is not unrealistic to assume that the optimization of the specification automata can begin hours or even days before the first model checking run. If such additional time is available, can it be used productively to obtain smaller automata that might then, once the verification starts, be used over and over again, when different systems (or, during debugging, multiple versions of the same system) are checked against the same properties?

In this paper, we present a new optimization technique that uses a modified SAT solver to search for a small automaton that accepts the same language as a given reference automaton $\mathcal{A}^+$. For this purpose, we encode a *candidate automaton* $\mathcal{A}$ with Boolean variables and check, after each decision made by the solver, whether the current partial variable valuation can still be completed to the representation of an automaton that is equivalent to $\mathcal{A}^+$. It is fairly simple to ensure that the candidate automaton only accepts words that are in the language of $\mathcal{A}^+$, because we can compute the complement $\mathcal{A}^-$ of $\mathcal{A}^+$ (if the starting point is an LTL formula, we simply run the LTL-to-Büchi translator on the negated formula). Then, $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{A}^+)$ corresponds to $\mathcal{L}(\mathcal{A}^-) \cap \mathcal{L}(\mathcal{A}) = \emptyset$, which is easy to check. The challenge, however, is to efficiently check $\mathcal{L}(\mathcal{A}^+) \subseteq \mathcal{L}(\mathcal{A})$, because here the complement would have to be recomputed for every candidate automaton.

We introduce *bounded* language containment, an approximative version of language containment, whose precision and computational cost can be fine-tuned so that the check runs in limited time. Bounded language containment between two Büchi automata $\mathcal{A}$ and $\mathcal{A}'$ requires that there exists a constant $b \in \mathbb{N}$ such that for each accepting run π of $\mathcal{A}$ on some word there is a run π' of $\mathcal{A}'$ on the same word such that the number of visits to accepting states in π between two visits of accepting states in π' is bounded by b. We can thus fine-tune the precision of bounded language containment by choosing different values for the bound: higher values result in greater precision, lower values in faster computation. No matter how we choose the bound, however, bounded language containment is always a sound approximation of language containment.

A SAT-based search for the smallest Büchi automaton that accepts the same language as a given reference automaton is slower than applying standard optimization techniques directly on the reference automaton. On automata from LTL-to-Büchi benchmarks [16, 8, 6], our prototype implementation often runs for several hours. By comparison, it usually only takes seconds to compute some reference automaton. However, the resulting size reduction is remarkable: even for automata obtained by the best currently available LTL-to-Büchi translators, `spot` [5] and `ltl2ba` [10], using various parameter settings on standard benchmarks for LTL-to-Büchi translation, our method improves in 43 out of 94 cases over the smallest automaton found by the LTL-to-Büchi tools, saving as many as 22 out of 28 states in one benchmark.

The remainder of the paper is structured as follows. In the following section, we review preliminaries on Büchi automata and SAT solving. In Section 3, we present our new algorithm for checking language equivalence based on bounded language containment. In Section 4, we integrate this algorithm into the SAT-based search for a small automaton that accepts the same language as a given reference automaton. Our experimental results are reported in Section 5. We conclude the paper in Section 6 with a discussion of the benefits and limitations of our approach in comparison to precise language containment and simulation-based approaches.

2 Preliminaries

2.1 Büchi Automata

A *Büchi automaton* $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ consists of a finite set of states Q, a finite alphabet Σ, a transition relation $\delta \subseteq Q \times \Sigma \times Q$ with a designated initial state $q_0 \in Q$, and a set of accepting states $F \subseteq Q$. We refer to the number of states in an automaton as its *size*.

A word over an alphabet Σ is an infinite sequence $w = w_0 w_1 \ldots \in \Sigma^\omega$. A *run* of $\mathcal{A}$ on an infinite word $w = w_0 w_1 \ldots \in \Sigma^\omega$ is an infinite sequence $\pi = \pi_0 \pi_1 \ldots \in Q^\omega$ of states where $\pi_0 = q_0$ and for all $i \in \mathbb{N}_0$, $(\pi_i, w_i, \pi_{i+1}) \in \delta$. A run π is *accepting* iff $\inf(\pi) \cap F \neq \emptyset$, where $\inf(\pi)$ denotes the set of states that occur infinitely often in π. The automaton is *without dead-ends* if every finite prefix of a run can be extended to an accepting run (possibly on a different input word). A word w is accepted by $\mathcal{A}$ iff there exists an accepting run for it. We denote the set of runs of $\mathcal{A}$ on w by $\mathcal{R}(\mathcal{A}, w)$, and the set of accepting runs by $\mathcal{R}_F(\mathcal{A}, w)$. For convenience, we extend the definition of $\mathcal{R}(\mathcal{A}, w)$ to finite prefixes $w \in \Sigma^*$ in the natural way. The set of accepted words is called the *language* $\mathcal{L}(\mathcal{A})$ of the automaton. Two automata $\mathcal{A}, \mathcal{A}'$ are *equivalent* if $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$.

An important application of Büchi automata is model checking for specifications given in *linear-time temporal logic* (LTL). The models of an LTL formula are infinite words over the alphabet $\Sigma = 2^{\mathsf{AP}}$, where AP is a fixed set of atomic propositions. In analogy to Büchi automata, we call the set of words that satisfy an LTL formula ψ the language $\mathcal{L}(\psi)$ of ψ. Model checkers like SPIN use translation algorithms that construct for a given LTL formula ψ a Büchi automaton such that $\mathcal{L}(\psi) = \mathcal{L}(\mathcal{A})$. Several such algorithms are described in the literature (cf. [16, 8, 10, 11]). For a comprehensive introduction to LTL and LTL model checking we refer the reader to standard textbooks on computer-aided verification (cf. [1]).

2.2 SAT Solving

Satisfiability (SAT) is the problem of determining if the variables of a given Boolean formula can be assigned in such a way that the formula evaluates to true. A SAT instance is given as a set of Boolean variables V and a set of clauses C, all of which are of the form $l_1 \vee \ldots \vee l_n$ for some set of literals $l_1, \ldots, l_n$ with $l_i \in V \cup \{\neg v : v \in V\}$ for all $1 \leq i \leq n$. An assignment of the variables to the values in $\mathbb{B} = \{\mathbf{false}, \mathbf{true}\}$ is called valid for $\langle V, C \rangle$ if it leads to the satisfaction of all clauses.

The algorithm presented in this paper is based on *search-based* SAT solving. Search-based SAT solvers maintain a *partial valuation* $V \to \{\mathbf{false}, \mathbf{true}, \bot\}$

of the variables by assigning to every variable either a truth value in $\mathbb{B}$ or the value $\bot$, indicating that no truth value has been decided for the variable yet. We call a partial evaluation $z' \in (V \to \{\mathbf{false}, \mathbf{true}, \bot\})$ an *extension* of a partial valuation $z \in (V \to \{\mathbf{false}, \mathbf{true}, \bot\})$ if for all $v \in V$ with $z(v) \in \mathbb{B}$, we have $z'(v) = z(v)$. A partial valuation z' is a *completion* of some partial valuation z if z' is an extension of z and every extension of z' is identical to z'.

In every step of the computation, the solver checks if the current decisions already make some clause unsatisfied, i.e., if a partial valuation has been reached where no completion of satisfies the clause. In this case, we say that a *conflict* has occurred and the solver *back-tracks* some of the decisions already made. Then, modern solvers analyze the cause of the conflict and store a learned clause that prevents the decisions that lead to the conflict from being made again.

SAT solving has been extended with *non-clausal constraints*, as they occur, for example, in *satisfiability modulo theory* solving [2]. The SAT instance is now given as a tuple $\langle V, C, N \rangle$, where the set of variables V and the set of clauses C is defined as before. Additionally, the set of non-clausal constraints $N \subseteq (V \to \{\mathbf{false}, \mathbf{true}, \bot\}) \to \mathbb{B}$ consists of functions that map partial valuations to a truth value indicating whether the partial valuation has a completion satisfying the constraint. Consequently, for a non-clausal constraint $f \in N$ and two partial valuations z, z', where z' is an extension of z, $f(z') = \mathbf{true}$ implies $f(z) = \mathbf{true}$. For SAT with non-clausal constraints, the solver checks, after each decision and for each non-clausal and clausal constraint, whether the current partial valuation can be extended in such a way that the constraint is satisfied. For more background on SAT solving, we refer the interested reader to a recent handbook [4].

3 Checking Language Equivalence

The key component of our minimization algorithm is the efficient test whether some candidate automaton accepts the same language as the reference automaton. As discussed in the introduction, we use a precise test to check whether the language of the candidate automaton is contained in the language of the reference automaton, and an approximative test, called *bounded* language containment, to check the opposite direction. These two tests are discussed in the following two subsections.

3.1 Precise Language Containment

The standard way to check if the language of an automaton $\mathcal{A}$ is contained in the language of an automaton $\mathcal{A}'$ is to check whether the intersection of $\mathcal{L}(\mathcal{A})$ with the complement of $\mathcal{L}(\mathcal{A}')$ is empty: $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{A}')$ iff $\mathcal{L}(\mathcal{A}) \cap (\Sigma^\omega \setminus \mathcal{L}(\mathcal{A}')) = \emptyset$.

The drawback of this approach is that the complementation of $\mathcal{A}'$ is expensive: the number of states in the complement automaton is exponential in the number of states of $\mathcal{A}$. In our construction, we use precise language containment for checking whether the language of a candidate automaton $\mathcal{A}$ is contained in

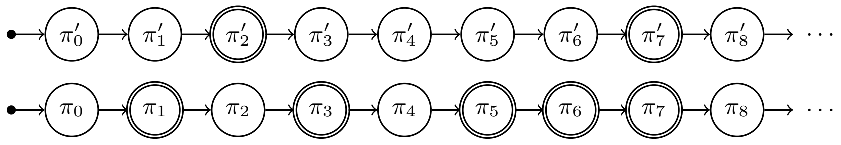


Fig. 2. A pair of runs with an acceptance lag of (at least) 3. The figure shows (prefixes of) runs π and π' of two Büchi automata on the same input word. Visits to accepting states are depicted as double circles. The acceptance lag between π and π' is at least 3, because π has 3 visits to accepting states between the two visits to accepting states of π' in positions 2 and 7.

the language of the reference automaton $\mathcal{A}^+$, but not for the opposite direction. In this way, only a single complementation, of the reference automaton $\mathcal{A}^+$ into its complement $\mathcal{A}^-$, is required. Furthermore, if $\mathcal{A}^+$ was obtained from an LTL formula ψ, we obtain $\mathcal{A}^-$ simply by translating the LTL formula $\neg\psi$ to a Büchi automaton.

3.2 Bounded Language Containment

Bounded language containment is an efficient approximative check if the language of an automaton $\mathcal{A}$ is contained in the language of an automaton $\mathcal{A}'$. In addition to the standard language containment condition, that for every word $w \in \Sigma^\omega$ and accepting run $\pi \in \mathcal{R}_F(\mathcal{A}, w)$ in $\mathcal{A}$, there must exist some accepting run $\pi' \in \mathcal{R}_F(\mathcal{A}', w)$ in $\mathcal{A}'$, we require that the number of visits to accepting states in π between two visits to accepting states in π' is bounded by some constant. Formally, the *acceptance lag* between a run $\pi = \pi_0\pi_1\ldots$ of $\mathcal{A}$ and a run $\pi' = \pi'_0\pi'_1\ldots$ of $\mathcal{A}'$ is defined as follows:

$$\begin{aligned}\operatorname{lag}(\pi, \pi') = \max\{ & j \in \mathbb{N}_0 : \exists x_1, \ldots, x_j \in \mathbb{N}_0 : (\forall 1 \le i < j : x_i < x_{i+1}) \\ & \wedge (\forall 1 \le i \le j : \pi_{x_i} \in F) \wedge (\forall x_1 \le i \le x_j : \pi'_i \notin F')\}\end{aligned}$$

In the example shown in Figure 2, the acceptance lag is (at least) 3, because π has 3 visits to accepting states (in positions 3, 5, and 6) between the two visits to accepting states of π' in positions 2 and 7.

Clearly, if π is an accepting run and the lag between π and π' is bounded by some constant, then π' is also accepting. Furthermore, if we have a word w that is in the language of $\mathcal{A}$ but not in the language of $\mathcal{A}'$, then the lag between some accepting run of $\mathcal{A}$ on w and an arbitrary run of $\mathcal{A}'$ on w is unbounded: any run of $\mathcal{A}'$ on w is rejecting and, hence, visits the accepting states only finitely often. Thus, if we fix some bound $b \in \mathbb{N}$ and observe that for every word w and every run $\pi \in \mathcal{R}_F(\mathcal{A}, w)$, the lag between π and some run $\pi' \in \mathcal{R}(\mathcal{A}', w)$ is at most b, we can conclude that indeed $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{A}')$.

This observation is the motivation for the definition of bounded language containment, which simultaneously checks the same bound for all words in Σ^ω. For a bound $b \in \mathbb{N}$, we say that the language of $\mathcal{A}$ is *b-bounded contained* in the language of $\mathcal{A}'$, denoted by $\mathcal{L}(\mathcal{A}) \subseteq_b \mathcal{L}(\mathcal{A}')$, if, for every word $w \in \Sigma^\omega$ and every $\pi \in \mathcal{R}_F(\mathcal{A}, w)$, there exists some $\pi' \in \mathcal{R}(\mathcal{A}', w)$ such that $\text{lag}(\pi, \pi') \leq b$. We use bounded language containment as a conservative approximation of language containment: for every $b \in \mathbb{N}$, $\mathcal{L}(\mathcal{A}) \subseteq_b \mathcal{L}(\mathcal{A}')$ implies $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{A}')$.

The idea of the bounded language containment checking algorithm we describe next is to encode bounded language containment as a graph reachability problem. The *lag-checking graph* branches according to the possible runs of $\mathcal{A}$ and keeps track of the possible runs of $\mathcal{A}'$ on the same input word. For this purpose, the vertices of the lag-checking graph contain a counter value $f(q')$ for each state q' of $\mathcal{A}'$ that indicates how many visits to accepting states in $\mathcal{A}$ (without visiting accepting states in $\mathcal{A}'$) are left before the paths through q' will exceed the bound on the acceptance lag.

Definition 1. *For two Büchi automata $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ and $\mathcal{A}' = (Q', \Sigma, \delta', q_0', F')$, and a bound $b \in \mathbb{N}$, the* lag-checking graph *$\mathcal{G}(\mathcal{A}, \mathcal{A}', b) = \langle V, E \rangle$ consists of the following set V of vertices and set $E \subseteq V \times \Sigma \times V$ of labelled edges:*

- *$V = Q \times (Q' \to \{0, \ldots, b+1\})$*
- *For all $(q_1, f_1), (q_2, f_2) \in V$ with $q_2 \in F$, and for all $x \in \Sigma$, we have $((q_1, f_1), x, (q_2, f_2)) \in E$ if and only if $(q_1, x, q_2) \in \delta$ and*
  - *for all $q_2' \in F'$: if there exist some $q_1' \in Q'$ with $f_1(q_1') > 0$ such that $(q_1', x, q_2') \in \delta'$, then $f_2(q_2') = b + 1$, otherwise $f_2(q_2') = 0$;*
  - *for all $q_2' \notin F'$: $f_2(q_2') = \max(\{f_1(q_1') - 1 : q_1' \in Q', (q_1', x, q_2') \in \delta'\} \cup \{0\})$.*
- *For all $(q_1, f_1), (q_2, f_2) \in V$ with $q_2 \notin F$, and for all $x \in \Sigma$, we have $((q_1, f_1), x, (q_2, f_2)) \in E$ if and only if $(q_1, x, q_2) \in \delta$ and*
  - *for all $q_2' \in F'$: if there exists some $q_1' \in Q'$ with $f(q_1') > 0$ such that $(q_1', x, q_2') \in \delta'$, then $f_2(q_2') = b + 1$, otherwise $f_2(q_2') = 0$;*
  - *for all $q_2' \notin F'$: $f_2(q_2') = \max(\{f_1(q_1') : q_1' \in Q', (q_1', x, q_2') \in \delta'\})$.*

If $q_0 \notin F$ or $q_0' \in F$, we call the vertex (q_0, f_0), where $f_0(q_0') = b + 1$ and $f_0(q') = 0$ for all $q' \in Q' \setminus \{q_0'\}$, the initial vertex; otherwise, the vertex (q_0, f_0), where $f_0(q_0') = b$ and $f_0(q') = 0$ for all $q' \in Q' \setminus \{q_0'\}$, is the initial vertex. Additionally, we call the vertices (q, f), where $f(q') = 0$ for all $q' \in Q'$, the final vertices.

Figure 3 shows, as an example, the lag-checking graph $\mathcal{G}(\mathcal{A}, \mathcal{A}', 1)$ for the automata $\mathcal{A}$ and $\mathcal{A}'$ from Figure 1 and bound 1. Since every reachable vertex has some non-zero counter, there exists, for every run on $\mathcal{A}$, a run of $\mathcal{A}'$ on the same input word such that the acceptance lag is bounded by 1. We formalize the meaning of the node labels in the following lemmas.

Lemma 2. *Let $w = w_0 \ldots w_{k-1} \in \Sigma^*$ be a finite word, $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ and $\mathcal{A}' = (Q', \Sigma, \delta', q_0', F')$ be two Büchi automata, $\pi = \pi_0 \ldots \pi_k$ be a prefix of a run of $\mathcal{A}$, and $b > 0$ be some bound. For any number $c > 0$ and state $q' \in Q'$,*

- *if there exists some path $(q_0, f_0) \xrightarrow{w_0} (q_1, f_1) \xrightarrow{w_1} \ldots \xrightarrow{w_{k-1}} (q_k, f_k)$ in $\mathcal{G}(\mathcal{A}, \mathcal{A}', b)$ from the initial vertex (q_0, f_0), such that $q_i = \pi_i$, for all $i \in \{0, \ldots, k\}$, and $f_k(q') = c$,*

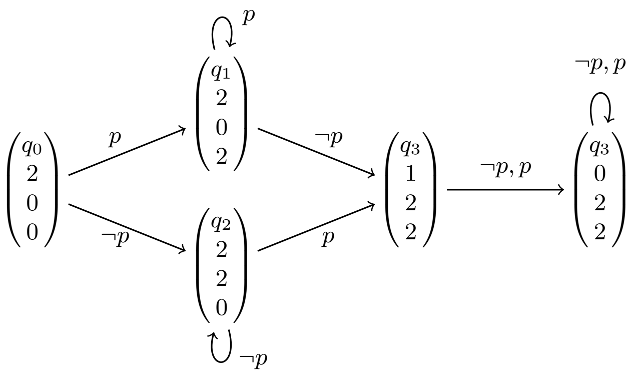

p
$\neg p, p$
$\begin{pmatrix} q_1 \\ 2 \\ 0 \\ 2 \end{pmatrix}$
p
$\neg p$
$\begin{pmatrix} q_0 \\ 2 \\ 0 \\ 0 \end{pmatrix}$
$\begin{pmatrix} q_3 \\ 1 \\ 2 \\ 2 \end{pmatrix}$
$\neg p, p$
$\begin{pmatrix} q_3 \\ 0 \\ 2 \\ 2 \end{pmatrix}$
$\neg p$
p
$\begin{pmatrix} q_2 \\ 2 \\ 2 \\ 0 \end{pmatrix}$
$\neg p$

Fig. 3. Example lag-checking graph that shows that language of the automaton depicted on the left in Figure 1 is 1-bounded contained in the language of the automaton depicted on the right in Figure 1. In each vertex (q, f) of the lag-checking graph, the values of q, $f(q'_0)$, $f(q'_1)$ and $f(q'_2)$ are shown (from top to bottom). No final vertex (i.e., no vertex (q, f) with $f(q') = 0$ for all $q' \in Q'$) is reachable.

– *then there exists a run prefix* $\pi' = \pi'_0 \ldots \pi'_k$ *in* $\mathcal{R}(\mathcal{A}', w)$ *with* $\pi'_k = q'$ *such that* $\text{lag}(\pi, \pi') \leq b$ *and* $b - c + 1$ *visits to* F *have occurred along* π *after the last visit to an accepting state in* π'.

Lemma 3. *Let* $w = w_0 \ldots w_{k-1} \in \Sigma^*$ *be a finite word,* $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ *and* $\mathcal{A}' = (Q', \Sigma, \delta', q'_0, F')$ *be two Büchi automata,* $\pi = \pi_0 \ldots \pi_k$ *be a prefix of a run of* $\mathcal{A}$, *and* $b > 0$ *be some bound. For any number* $c > 0$ *and state* $q' \in Q'$,

– *if there exists a run prefix* $\pi' = \pi'_0 \ldots \pi'_k$ *in* $\mathcal{R}(\mathcal{A}', w)$ *with* $\pi'_k = q'$ *such that* $\text{lag}(\pi, \pi') \leq b$ *and* $b - c + 1$ *visits to* F *have occurred along* π *after the last visit to an accepting state in* π',
– *then there exists some path* $(q_0, f_0) \xrightarrow{w_0} (q_1, f_1) \xrightarrow{w_1} \ldots \xrightarrow{w_{k-1}} (q_k, f_k)$ *in* $\mathcal{G}(\mathcal{A}, \mathcal{A}', b)$ *from the initial vertex* (q_0, f_0), *such that* $q_i = \pi_i$, *for all* $i \in \{0, \ldots, k\}$, *and* $f_k(q') \geq c$.

The lemmas are easy to prove by induction over the length of w, π and π' and a case split on the four possible combinations of whether $\pi_k \in F$ and $\pi'_k \in F'$ hold or not. We use the two lemmas to reduce the bounded language containment check to a reachability property of the lag-checking graph:

Theorem 4. *Let* $\mathcal{A}$ *and* $\mathcal{A}'$ *be two Büchi automata such that* $\mathcal{A}$ *has no dead-ends, and let* $b \in \mathbb{N}$ *be some bound. The following two conditions are equivalent:*

1. $\mathcal{L}(\mathcal{A}) \subseteq_b \mathcal{L}(\mathcal{A}')$;
2. *in the lag-checking graph* $\mathcal{G}(\mathcal{A}, \mathcal{A}', b)$, *no final vertex is reachable from the initial node.*

The reachability of the final nodes can be checked by a simple depth-first or breadth-first graph traversal.

4 SAT-based Minimization of Büchi Automata

We now describe a SAT-based algorithm for finding a small Büchi automaton that accepts the same language as a given reference automaton $\mathcal{A}^+$. We use a SAT solver to determine if, for a given number of states n, there is some automaton $\mathcal{A}$ that is equivalent to $\mathcal{A}^+$. For this purpose, we encode the candidate automata symbolically using Boolean variables and search for a valuation that corresponds to an automaton that passes the equivalence check defined in the previous section. We start by defining, in the following subsection, the Boolean encoding of the candidate automata. In Sections 4.2 and 4.3 we adapt the language containment tests from the previous section to this setting.

4.1 Boolean Encodings of Büchi Automata

We give a Boolean encoding for a candidate automaton $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ over a given alphabet Σ and with a given number of states $n = |Q|$. Without loss of generality, we assume $Q = \{1, \ldots, n\}$ and $q_0 = 1$. It remains to encode the transition relation δ and the set F of accepting states.

For every pair $q, q' \in Q$ of states and input letter $s \in \Sigma$, we define a boolean variable $\langle q, s, q' \rangle_\delta$ indicating whether $(q, s, q') \in \delta'$. Likewise, for all states $q \in Q$, a boolean variable $\langle q \rangle_F$ is used for representing whether the state is accepting or not.

Checking if there exists an automaton $\mathcal{A}$ with n states and $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}^+)$ can be done by iterating over all possible transition relations δ and sets of final states F. For every combination of δ and F, we check whether the language of $\mathcal{A}$ with these values of δ and F is the same as the language of $\mathcal{A}^+$. For our encoding, the overall search space thus has a size of $2^{n^2|\Sigma|+n}$. We can assume that $\mathcal{A}^+$ has more than n states as otherwise the problem is trivial to solve (by taking $\mathcal{A} = \mathcal{A}^+$).

In the remainder of this section, we describe how to modify a SAT solver to search for smaller equivalent Büchi automata over this Boolean encoding. For this purpose we adapt the language equivalence check developed in the previous section to work on *partially specified automata*, given by the partial valuation of the variables $\langle \cdot \rangle_\delta$ and $\langle \cdot \rangle_F$ provided by the SAT solver during the search.

This allows the SAT solver to recognize conflicts early. A commonly occurring situation is, for example, that the candidate automaton has a self-loop in an accepting initial state and, as a result, accepts words that are not in the language of $\mathcal{A}^+$. In this case, the decision procedure should be able to identify this situation already after only the two corresponding bits have been set of **true**.

We split the language equivalence $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}^+)$ into its two language containment relations. For $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{A}^+)$, we adapt the precise language containment check from Section 3.1, for $\mathcal{L}(\mathcal{A}^+) \subseteq \mathcal{L}(\mathcal{A})$ the bounded language containment check from Section 3.2. The two constructions are explained in the following subsections.

4.2 Checking $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{A}^+)$

As discussed in Section 3.1, we reduce $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{A}^+)$ to checking whether the intersection of $\mathcal{L}(\mathcal{A})$ with the complement of $\mathcal{L}(\mathcal{A}^+)$ is empty.

During the search for a satisfying variable valuation, we perform the test already on partially specified automata, i.e., when for some transitions it is not (yet) known whether they are contained in the transition relation or not, and for some states it is not (yet) known whether they are contained in the set of accepting states. Thus, we need to be able to check if the candidate automaton can be completed in a way such that (bounded) language containment holds.

In order to check language containment for a partially specified candidate automaton, we interpret the "undecided" value $\perp$ in a partial valuation as **false**. Since this eliminates any transitions and accepting states that are not required by the partial valuation, we thus obtain the candidate automaton with the *least* language that is compatible with the decisions made so far.

If the intersected language is non-empty, we also provide the SAT solver with a conflict clause. For this purpose, the emptiness check on the intersection automaton, which searches for a lasso path from the initial state to some loop that contains an accepting state, annotates every state in the intersection automaton with the predecessor state encountered during the search. When an accepting lasso is found, this annotation is used to traverse the lasso backwards. By collecting the negations of the literals in the SAT instance corresponding to the transitions on the lasso and the literal for the accepting state, we extract a conflict clause, which is used by the learning scheme of the SAT solver to avoid a repetition of the decisions that have allowed the language of the candidate automaton to become too large.

4.3 Checking $\mathcal{L}(\mathcal{A}^+) \subseteq \mathcal{L}(\mathcal{A})$

We check $\mathcal{L}(\mathcal{A}^+) \subseteq \mathcal{L}(\mathcal{A})$ using the bounded language containment test described in Section 3.2. In order to apply the test to partially specified candidate automata, we interpret the "undecided" value $\perp$ in a partial valuation as **true**, and thus obtain the candidate automaton with the *greatest* language that is compatible with the decisions made so far.

If the bounded language containment test $\mathcal{L}(\mathcal{A}^+) \subseteq_b \mathcal{L}(\mathcal{A})$ fails, we again supply the SAT solver with a cause for the conflict in order to benefit from its learning scheme. The test fails when a final vertex is reachable in the lag-checking graph $\mathcal{G}(\mathcal{A}^+, \mathcal{A}, b)$. We collect the labels $x \in \Sigma$ observed along the path from the initial vertex to some final vertex and report all SAT variables set to **false** corresponding to these alphabet symbols, along with all variables for final states that are set to **false**.

4.4 Symmetry Breaking

The concept of symmetry breaking has been identified to be indispensable for the efficient solution of many SAT problems [15]. Symmetry breaking prunes the search space of a SAT solver by adding constraints that remove some solutions that are isomorphic to others which are not pruned away. In the case of Büchi automaton minimization, for example, swapping two states in the automaton results in an automaton that is isomorphic to the original one.

For the purposes of this work, we break symmetry only partially as it has been observed that this is often faster than performing total symmetry breaking [15], where in the pruned state space, only one representative of each equivalence class of the automata equivalent under isomorphism remains. This is done by choosing some order over the variables and adding a (non-clausal) constraint to the SAT problem that for n being the number of states of the candidate automaton, for every $i \in \{1, \ldots, n-1\}$, swapping the ith and $(i+1)$th state does not result in an automaton that is lexicographically smaller with respect to this bit order.

5 Experimental Evaluation[1]

We have evaluated the automaton minimization approach presented in this paper on three benchmark sets from the literature, which have previously been used to evaluate and compare LTL-to-Büchi translators:

– The 27 specifications from the paper "Efficient Büchi Automata from LTL Formulae" by Fabio Somenzi and Roderick Bloem [16].
– The 12 specifications from the paper "Optimizing Büchi Automata" by Kousha Etessami and Gerard J. Holzmann [8].
– 55 LTL properties built from the specification patterns in the paper "Property Specification Patterns for Finite-state Verification" by Matthew B. Dwyer, George S. Avrunin and James C. Corbett [6].

We have implemented a single-threaded prototype tool on top of the SAT solver `minisat v.1.12b` [7]. Our tool reads in a reference automaton $\mathcal{A}^+$ and its complement $\mathcal{A}^-$ over some fixed alphabet Σ (using SPIN never-claim syntax) and checks, for some given bound $b \in \mathbb{N}$ and size $n \in \mathbb{N}$, whether there exists a Büchi automaton $\mathcal{A}$ with n states such that $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{A}^+)$ and $\mathcal{L}(\mathcal{A}^+) \subseteq_b \mathcal{L}(\mathcal{A})$ (assuming that $\mathcal{L}(\mathcal{A}^+) = \Sigma^\omega \setminus \mathcal{L}(\mathcal{A}^-)$). All benchmarks were performed on a computer with AMD Opteron 2.8Ghz processors running Linux. In our experiments, we have used bounds between 1 and 8.

Our goal has been to start with Büchi automata that have already been optimized by the best currently available methods. Starting with the LTL formulas from the benchmark suites, we therefore applied the two best currently available LTL-to-Büchi converters [14], namely `spot v.0.5a` [5] and `ltl2ba v.1.1` [10] to compute the automata $\mathcal{A}^+$ and $\mathcal{A}^-$. The `spot` tool combines several approaches and optimizations for the conversion process in one program; we applied all 15 different parameter combinations and took for every specification the

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

Table 1. Running times (in seconds) of our prototype tool for the specification $(\mathsf{G}\neg s) \vee \mathsf{F}(s \wedge (\neg r \mathsf{U}(t \vee \mathsf{G}\neg r)))$, inducing automata sizes of $|\mathcal{A}^+| = 6$ and $|\mathcal{A}^-| = 3$. Gray table cells represent cases in which the solver found out that the instance is satisfiable, i.e., some smaller automaton has been found. The respective tables for all 94 benchmarks can be found at `http://react.cs.uni-saarland.de/tools/nbwminimizer`.

# states / bound	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.02	0.0	0.01
3	0.01	0.03	0.05	0.1	0.1	0.1	0.12	0.11
4	0.52	1.15	2.21	1.33	1.07	2.35	2.74	2.05
5	0.3	1.03	27.28	10.24	56.41	59.21	80.2	80.1

smallest automata encountered for some tool/parameter combination. In total, this scheme resulted in $94 \cdot 16 \cdot 2$ calls to LTL-to-Büchi converters. In two of these cases, running `spot` required more than 5 minutes due to a expensive optimization. In these cases, we aborted the run and took the best automaton found by some other run instead. All in all, the computation of the Büchi automata took about 60 minutes on the computer mentioned above.

Table 1 shows the running times of our prototype tool on a typical (but rather small) specification from the third benchmark set. For some smaller values of the bound and number of states in the candidate automaton, the solver ran too quickly to measure its running time with the Linux `time` utility. On the other hand, needlessly large values of n and b can delay the termination of the SAT solver unnecessarily. Thus, for a meaningful application of the solver in practice, an *evaluation scheme* is necessary, which fixes how the solver runs for the individual bound and target state number values are combined to an overall Büchi automaton size reduction program. We propose two such schemes here:

- **Scheme A:** The solver is applied repeatedly to the state number/bound pairs $(1,1), (1,2), \ldots, (1,8), (2,1), (2,2), \ldots$ until we obtain the result that the SAT instance is satisfied (so a smaller automaton is found) or until the results for all state number/bound pairs have been computed.
- **Scheme B:** We assume that for all state number/bound pairs, the SAT solving instances are started in parallel (taking for granted that we have sufficiently many computers at hand). Once the minimal state number (for the chosen maximal bound of 8) can be deduced from the results obtained so far, all remaining SAT solving runs are stopped. Here, the computation time is defined to be the time after which the solver runs are stopped.

In both cases, we have chosen a timeout of 12 hours and a memory limit of 2 GB for the individual runs of the SAT solver. Table 2 contains an overview of the results for these two schemes, grouped by the benchmark sets. In all of the cases, the smallest automata we found have been obtained already with a bound of 2. Furthermore, in most cases where a smaller Büchi automaton was found, this was already the case with a bound of 1. Additionally, for only 8 out of the

Table 2. Overview table of the experimental evaluation, grouped by benchmark suites.

	[16]	[8]	[6]
# Automata in benchmark suites	27	12	55
# Cases for which a smaller automaton was found	7	0	36
# Cases for which no smaller automaton was found	20	10	15
# Cases for which a smaller automaton was found but it is not known if it is the smallest one (w.r.t. the maximum bound of 8)	2	0	2
# Cases for which it is unknown if there exists a smaller automaton	0	2	4
Maximum size saving:	3 out of 5	−/−	22 out of 28
Average size saving:	7.28%	0.0%	24.0%
Mean computation times (Scheme A):	85669.87 s	83711.1 s	194558.8 s
Mean computation times (Scheme B):	4454.274 s	7203.017 s	4850.677 s
Minimum ratio between $\mathcal{A}^+$ and $\mathcal{A}^-$:	1/2	2/3	1/2
Median ratio between $\mathcal{A}^+$ and $\mathcal{A}^-$:	2/2	2/2	5/3
Maximum ratio between $\mathcal{A}^+$ and $\mathcal{A}^-$:	5/4	6/6	28/5

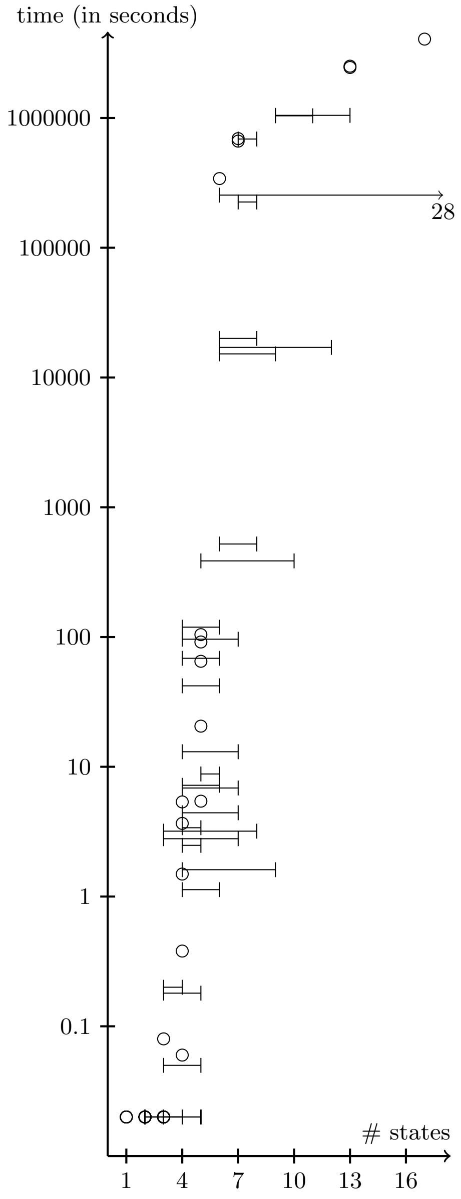
94 specifications considered, the maximum time of 12 hours per SAT solver run was insufficient to decide whether the input automaton was already the smallest equivalent one (with respect to the maximum bound of 8), which shows that the technique presented in this paper is indeed useful.

Finally, we discuss the scalability of our approach. Figure 4 correlates the original and reduced sizes of the automata for the specifications considered with the running times of the reduction process using the schemes described above. It can be seen that the running time of our technique is roughly exponential in the number of states. However, the figure shows that with this investment in terms of computation time, significant size savings are possible.

6 Discussion

In this paper, we have presented a new approach for the optimization of Büchi automata that allows for significant further size reductions over standard techniques when more time can be invested in the minimization of the automaton. The new approach is based on a combination of bounded language containment checking, which allows us to fine-tune the precision and efficiency of the check, and SAT solving, which allows us to prune large parts of the search space for smaller automata already in the early solving stages. We conclude the paper by discussing the limits of the approach and its relation to simulation-based optimization methods.

A limitation of the approach is that bounded language containment is only an approximation of language containment. Figure 5 shows two equivalent au-



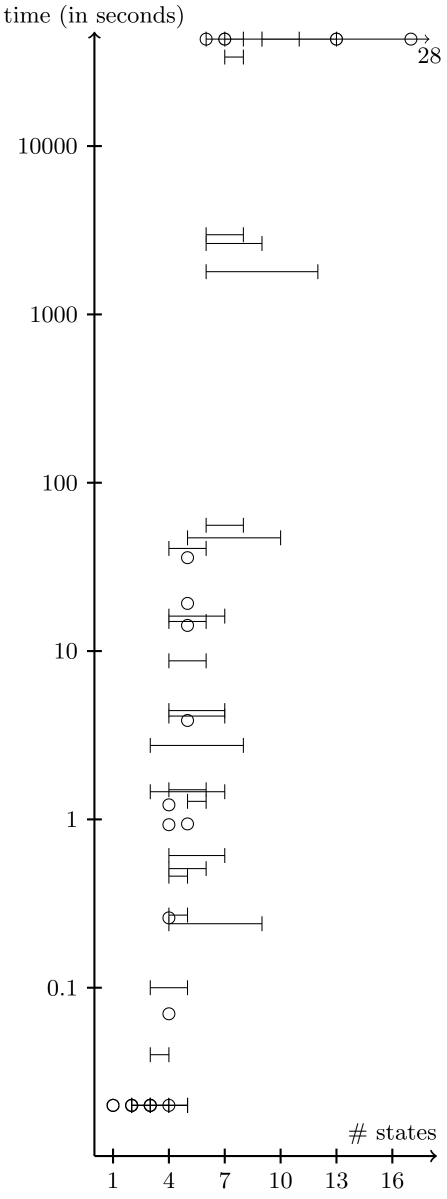


Fig. 4. Running times (in seconds, Y-axis) and automata sizes (in number of states, X axis) for the 95 specifications in our experimental evaluation. The left table corresponds to evaluation scheme A, the right table to scheme B. Automata whose sizes were not reduced by our technique are denoted by a circle, bars indicate the original and reduced sizes of the automata whenever a smaller automaton was found.

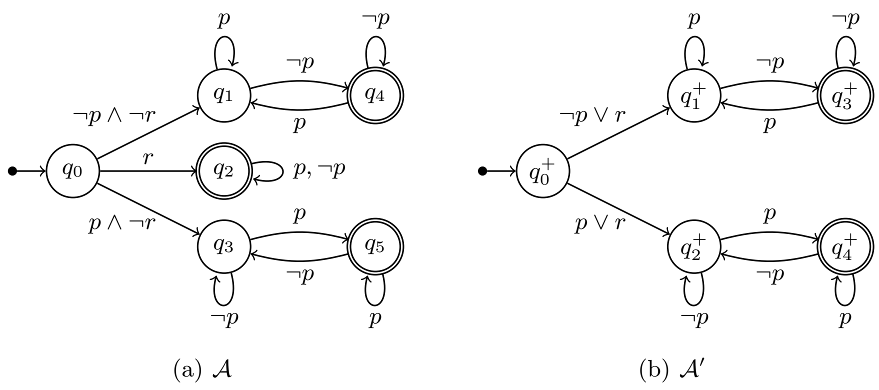


Fig. 5. Two Büchi automata, $\mathcal{A}$ and $\mathcal{A}'$ that are language equivalent, but not bounded language equivalent: $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$, but $\mathcal{L}(\mathcal{A}) \not\subseteq_b \mathcal{L}(\mathcal{A}')$ for every $b \in \mathbb{N}$. For the word $w = \{r\}\{p\}^{(b+1)}\emptyset^{(b+1)}\{p\}^\omega$, the acceptance lag between the unique accepting runs for this word is b, for every $b \in \mathbb{N}$.

tomata for which the bounded language containment check $\mathcal{L}(\mathcal{A}) \subseteq_b \mathcal{L}(\mathcal{A}')$ fails. Consider the word $w = \{r\}\{p\}^{(b+1)}\emptyset^{(b+1)}\{p\}^\omega$ for some value of $b \in \mathbb{N}$. Regardless of the actual choice of b, the word is clearly accepted by both automata. The acceptance lag between the unique accepting runs for this word is b. Thus, for all $b \in \mathbb{N}$, there exists a witness for the fact that $\mathcal{L}(\mathcal{A}) \not\subseteq_b \mathcal{L}(\mathcal{A}')$, showing that bounded language containment is sound but not complete.

In practice, of course, this limitation matters very little, since, in order to limit the running time, the test cannot be used with arbitrarily large bounds. In fact, our experimental evaluation shows that the approach is already very useful for small bounds. This success can be explained by the fact the bounded language containment check, which is the only part of the approach which can make it incomplete, has several nice properties. First of all, if the reference automaton is a safety automaton (a safety automaton is a Büchi automaton with only accepting states), then even for a bound of 1, we never miss a smaller safety automaton in the search process. This can easily be seen from the fact that the acceptance lag between an accepting run of the candidate automaton and an accepting run of the reference automaton is always 0 (as all states are accepting). Thus, our technique is complete for the minimization of safety automata, which is itself a PSPACE-complete problem.

Furthermore, the approach is complete with respect to previous approaches to the minimization of Büchi automata. In [9], the usage of *fair simulation* for approximating Büchi automaton language containment is discussed. In this setting, language containment can be proven by showing that a *simulation parity game* is winning for the *duplicator* player. From the structure of this parity game and the fact that parity games are memoryless determined, it can immediately be deduced that the duplicator player can only win if it is able to mimic an accepting run in the reference automaton $\mathcal{A}^+$ by some accepting run in the candidate automaton $\mathcal{A}$ such that the acceptance lag is below $|\mathcal{A}^+| \cdot |\mathcal{A}|$. Thus, by setting the bound in our approach to this value, we do not miss automata that would be found using fair bisimulation. Furthermore, in our technique, one direction of the language containment check is precise and by using SAT solving as the reasoning backbone, the limitations of bisimulation quotienting for fair simulation in the classical approaches are avoided.

Note that our approach *strictly* subsumes simulation-based methods (for fair, delayed and direct simulation). One such example is given in Figure 1: states q'_1 and q'_2 do not simulate state q_3. As a result, simulation-based methods cannot reduce the 4-state automaton $\mathcal{A}$ to the 3-state automaton $\mathcal{A}'$.

Acknowledgements

The authors want to thank Alexandre Duret-Lutz from the `spot` team for the support with the specifications and the converter.

References

1. Baier, C., Katoen, J.P.: Principles of Model Checking. MIT Press (2008)
2. Barrett, C., Sebastiani, R., Seshia, S., Tinelli, C.: Satisfiability Modulo Theories. [4] 825–885
3. Berry, G., Comon, H., Finkel, A., eds.: Computer Aided Verification, 13th International Conference, CAV 2001, Paris, France, July 18-22, 2001, Proceedings. In Berry, G., Comon, H., Finkel, A., eds.: CAV. Volume 2102 of Lecture Notes in Computer Science., Springer (2001)
4. Biere, A., Heule, M., van Maaren, H., Walsh, T., eds.: Handbook of Satisfiability. IOS Press (2009)
5. Duret-Lutz, A., Poitrenaud, D.: Spot: An extensible model checking library using transition-based generalized büchi automata. In DeGroot, D., Harrison, P.G., Wijshoff, H.A.G., Segall, Z., eds.: MASCOTS, IEEE Computer Society (2004) 76–83
6. Dwyer, M.B., Avrunin, G.S., Corbett, J.C.: Property specification patterns for finite-state verification. In Ardis, M., ed.: Proceedings of the 2nd Workshop on Formal Methods in Software Practice (FMSP'98), New York, ACM Press (March 1998) 7–15
7. Eén, N., Sörensson, N.: An extensible SAT-solver. In Giunchiglia, E., Tacchella, A., eds.: SAT. Volume 2919 of Lecture Notes in Computer Science., Springer (2003) 502–518
8. Etessami, K., Holzmann, G.J.: Optimizing Büchi automata. In Palamidessi, C., ed.: CONCUR. Volume 1877 of Lecture Notes in Computer Science., Springer (2000) 153–167
9. 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.: ICALP. Volume 2076 of Lecture Notes in Computer Science., Springer (2001) 694–707
10. Gastin, P., Oddoux, D.: Fast LTL to Büchi automata translation. [3] 53–65
11. Gerth, R., Peled, D., Vardi, M.Y., Wolper, P.: Simple on-the-fly automatic verification of linear temporal logic. In Dembinski, P., Sredniawa, M., eds.: PSTV. Volume 38 of IFIP Conference Proceedings., Chapman & Hall (1995) 3–18

12. Giannakopoulou, D., Lerda, F.: From states to transitions: Improving translation of LTL formulae to Büchi automata. In Peled, D., Vardi, M.Y., eds.: FORTE. Volume 2529 of Lecture Notes in Computer Science., Springer (2002) 308–326
13. Holzmann, G.: The Spin model checker: primer and reference manual. Addison-Wesley Professional (2003)
14. Rozier, K.Y., Vardi, M.Y.: LTL satisfiability checking. In Bosnacki, D., Edelkamp, S., eds.: SPIN. Volume 4595 of Lecture Notes in Computer Science., Springer (2007) 149–167
15. Sakallah, K.A.: Symmetry and Satisfiability. [4] 289–338
16. Somenzi, F., Bloem, R.: Efficient Büchi automata from LTL formulae. In Emerson, E.A., Sistla, A.P., eds.: CAV. Volume 1855 of Lecture Notes in Computer Science., Springer (2000) 248–263

A Proofs

A.1 Proofs of Lemma 2 and Lemma 3

For simplicity, for the following proofs, we define a function ψ over two traces in $\mathcal{A}$ and $\mathcal{A}'$ for all $n \in \mathbb{N}_0$, $\pi = \pi_0 \ldots \pi_n \in Q^n$ and $\pi' = \pi'_0 \ldots \pi'_n \in (Q')^n$ as follows:

$$\psi(\pi, \pi') = \max\{j \in \mathbb{N}_0 : \exists x_1, \ldots, x_j \in \mathbb{N}_0 : (\forall 1 \leq i < j : x_i < x_{i+1}) \\ \wedge (\forall 1 \leq i \leq j : \pi_{x_i} \in F) \wedge (\forall x_1 \leq i \leq n : \pi'_i \notin F')\}$$

Additionally, we define $\{a \mapsto b\}$ to denote the function mapping a to b and all other elements of some set containing a to the value 0.

Note that the definitions of the functions lag and ψ only differ in one point: only the parts in the prefix runs after the last visit to F' are taken into account. Thus, ψ formalizes the prose text in the claim of Lemma 2. We can thus restate the lemmas as follows:

Lemma 5. *Let $w = w_0 \ldots w_{k-1} \in \Sigma^*$ be a finite word, $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ and $\mathcal{A}' = (Q', \Sigma, \delta', q'_0, F')$ be two Büchi automata, $\pi = \pi_0 \ldots \pi_k$ be a prefix of a run of $\mathcal{A}$, and $b > 0$ be some bound. For any number $c > 0$ and state $q' \in Q'$,*

- *if there exists some path $(q_0, f_0) \xrightarrow{w_0} (q_1, f_1) \xrightarrow{w_1} \ldots \xrightarrow{w_{k-1}} (q_k, f_k)$ in $\mathcal{G}(\mathcal{A}, \mathcal{A}', b)$ from the initial vertex (q_0, f_0), such that $q_i = \pi_i$, for all $i \in \{0, \ldots, k\}$, and $f_k(q') = c$,*
- *then there exists a run prefix $\pi' = \pi'_0 \ldots \pi'_k$ in $\mathcal{R}(\mathcal{A}', w)$ with $\pi'_k = q'$ such that $\mathrm{lag}(\pi, \pi') \leq b$ and $\psi(\pi, \pi') = b - c + 1$.*

Proof. The proof is carried out by induction.

- **Induction basis:**
  - **Case $q_0 \in F \wedge q'_0 \notin F'$:** In this case, we have that the initial vertex in $\mathcal{G}(\ldots)$ is defined to be $(q_0, \{q'_0 \mapsto b\})$. Clearly, $\psi(q_0, q'_0) = 1$ and $\mathrm{lag}(q_0, q'_0) \leq 1 \leq b$.
  - **Case $q_0 \notin F \vee q'_0 \in F'$:** In this case, we have that the initial vertex in $\mathcal{G}(\ldots)$ is defined to be $(q_0, \{q'_0 \mapsto b + 1\})$. Clearly, $\psi(q_0, q'_0) = 0$ and $\mathrm{lag}(q_0, q'_0) = 0 < b$.
- **Induction step:** Assume that the claim holds for some k. Then we extend it to $k + 1$.
  - **Case $q_{k+1} \notin F^+ \wedge q' \notin F$:**
    - Assume that we have some path $(q_0, f_0) \xrightarrow{w_0} (q_1, f_1) \xrightarrow{w_1} \ldots \xrightarrow{w_k} (q_{k+1}, f_{k+1})$ given.
    - By the induction hypothesis, for all $q'_p \in Q$ and $c_p > 0$, we can deduce from the fact that $f_k(q'_p) = c_p$ that there exists some path $\pi' = \pi'_0 \ldots \pi'_k$ in $\mathcal{A}'$ with $\mathrm{lag}(q_0 \ldots q_k, \pi') \leq b$ for and $\psi(q_0 \ldots q_k, \pi') = b - c_p + 1$.

  * We have to find such a path $\tilde{\pi}'$ for q' and c from $(q_0, f_0) \xrightarrow{w_0} (q_1, f_1) \xrightarrow{w_1} \ldots \xrightarrow{w_k} (q_{k+1}, f_{k+1})$ under the assumption that $f_{k+1}(q') = c > 0$.
  * Since $q_{k+1} \notin F$ and $q' \notin F'$, we can assume that there exists some $q'_p \in Q'$ such that $(q'_p, w_k, q') \in \delta'$ and $f_k(q'_p) = c$ as otherwise we cannot have $(q_k, f_k) \xrightarrow{w_k} (q_{k+1}, f_{k+1})$ in $\mathcal{G}(\ldots)$.
  * Since $q_{k+1} \notin F$ and $q' \notin F'$, this means that the path $\tilde{\pi}' = \pi'_0 \ldots \pi'_k q'$ has $\text{lag}(q_0 \ldots q_{k+1}, \tilde{\pi}') \leq b$ for suitable $\pi'_0 \ldots \pi'_k$ (which must exist by the IH) and $\psi(q_0 \ldots q_{k+1}, \tilde{\pi}') = b - c + 1$.
- **Case** $q' \in F'$**:**
  * Assume that we have some path $(q_0, f_0) \xrightarrow{w_0} (q_1, f_1) \xrightarrow{w_1} \ldots \xrightarrow{w_k} (q_{k+1}, f_{k+1})$ given.
  * By the induction hypothesis, for all $q'_p \in Q$ and $c_p > 0$, we can deduce from the fact that $f_k(q'_p) = c_p$ that there exists some path $\pi' = \pi'_0 \ldots \pi'_k$ in $\mathcal{A}'$ with $\text{lag}(q_0 \ldots q_k, \pi') \leq b$ and $\psi(q_0 \ldots q_k, \pi') = b - c_p + 1$.
  * We have to find such a path $\tilde{\pi}'$ for q' and c from $(q_0, f_0) \xrightarrow{w_0} (q_1, f_1) \xrightarrow{w_1} \ldots \xrightarrow{w_k} (q_{k+1}, f_{k+1})$ under the assumption that $f_{k+1}(q') = c > 0$.
  * Since $c > 0$, by the construction of $\mathcal{G}(\ldots)$, there must exist some q'_p such that $f_k(q'_p) = c_p$ for some $c_p > 0$ and $(q'_p, w_k, q') \in \delta'$. Furthermore we know that in this case $c = b + 1$.
  * We can thus construct $\tilde{\pi}' = \pi'_0 \ldots \pi'_k q'$ for suitable $\pi'_0 \ldots \pi'_k$ (which must exist by the IH). Since $\text{lag}(q_0 \ldots q_k, \pi') \leq b$ and $q' \in F'$, we have $\text{lag}(q_0 \ldots q_{k+1}, \tilde{\pi}') \leq b$ and $\psi(q_0 \ldots q_{k+1}, \tilde{\pi}') = 0$.
- **Case** $q_{k+1} \in F \wedge q' \notin F'$**:**
  * Assume that we have some path $(q_0, f_0) \xrightarrow{w_0} (q_1, f_1) \xrightarrow{w_1} \ldots \xrightarrow{w_k} (q_{k+1}, f_{k+1})$ given.
  * By the induction hypothesis, for all $q'_p \in Q$ and $c_p > 0$, we can deduce from the fact that $f_k(q'_p) = c_p$ that there exists some path $\pi' = \pi'_0 \ldots \pi'_k$ in $\mathcal{A}'$ with $\text{lag}(q_0 \ldots q_k, \pi') \leq b$ and $\psi(q_0 \ldots q_k, \pi') = b - c_p + 1$.
  * We have to find such a path $\tilde{\pi}'$ for q' and c from $(q_0, f_0) \xrightarrow{w_0} (q_1, f_1) \xrightarrow{w_1} \ldots \xrightarrow{w_k} (q_{k+1}, f_{k+1})$ under the assumption that $f_{k+1}(q') = c > 0$.
  * Since $c > 0$ and due to the construction of $\mathcal{G}(\ldots)$, we know that there must exist some q'_p such that $f_k(q'_p) = c + 1$ and $(q'_p, w_k, q') \in \delta'$. As we know that $\text{lag}(q_0 \ldots q_k, \pi') \leq b$ and $\psi(q_0 \ldots q_k, \pi') = b - c + 2$, we know that $\text{lag}(q_0 \ldots q_k q_{k+1}, \tilde{\pi}') \leq b$ for $\tilde{\pi}' = \pi'_0 \ldots \pi'_k q'$ and $\psi(q_0 \ldots q_k q_{k+1}, \tilde{\pi}') = b - c + 1$ for suitable $\pi'_0 \ldots \pi'_k$ (which must exist by the IH).

Lemma 6. *Let $w = w_0 \ldots w_{k-1} \in \Sigma^*$ be a finite word, $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ and $\mathcal{A}' = (Q', \Sigma, \delta', q'_0, F')$ be two Büchi automata, $\pi = \pi_0 \ldots \pi_k$ be a prefix of a run of $\mathcal{A}$, and $b > 0$ be some bound. For any number $c > 0$ and state $q' \in Q'$,*

- *if there exists a run prefix $\pi' = \pi'_0 \dots \pi'_k$ in $\mathcal{R}(\mathcal{A}', w)$ with $\pi'_k = q'$ such that* $\mathrm{lag}(\pi, \pi') \leq b$ *and* $\psi(\pi, \pi') = b - c + 1$
- *then there exists some path $(q_0, f_0) \xrightarrow{w_0} (q_1, f_1) \xrightarrow{w_1} \dots \xrightarrow{w_{k-1}} (q_k, f_k)$ in $\mathcal{G}(\mathcal{A}, \mathcal{A}', b)$ from the initial vertex (q_0, f_0), such that $q_i = \pi_i$, for all $i \in \{0, \dots, k\}$, and $f_k(q') \geq c$.*

Proof. This proof is also carried out by induction.

- **Induction basis:**
  - **Case $q_0 \in F \wedge q'_0 \notin F'$:** Here, for $\mathcal{A}$ as well as $\mathcal{A}'$, there is only one path per automaton for the empty word, namely q_0 and q'_0, respectively. Obviously, $\mathrm{lag}(q_0, q'_0) \leq b$ for $b \geq 1$. Additionally, the initial vertex (q_0, f_0) has $f_0(q'_0) = b$ by the construction of $\mathcal{G}(\dots)$. As for $q_0 \in F \wedge q'_0 \notin F'$, $\psi(q_0, q'_0) = 1$, the claim holds.
  - **Case $q_0 \notin F \vee q'_0 \in F'$:** Here, for $\mathcal{A}$ as well as $\mathcal{A}'$, there is only one path per automaton for the empty word, namely q_0 and q'_0, respectively. Obviously, $\mathrm{lag}(q_0, q'_0) \leq b$ for $b \geq 1$. Additionally, the initial vertex (q_0, f_0) has $f_0(q'_0) = b + 1$ by the construction of $\mathcal{G}(\dots)$. As for $q_0 \notin F \vee q'_0 \in F$, we have $\psi(q_0, q'_0) = 0$, the claim holds.
- **Induction step:** By the induction hypothesis, we can assume that for every prefix path $\pi = \pi_0 \dots \pi_k$ in $\mathcal{R}(\mathcal{A}, w_0 \dots w_{k-1})$ and every prefix path $\pi' = \pi'_0 \dots \pi'_k$ in $\mathcal{R}(\mathcal{A}', w_0 \dots w_{k-1})$ with $\pi'_k = q'_p$ for some $q'_p \in Q'$, $\mathrm{lag}(\pi, \pi') \leq b$ and $\psi(\pi, \pi') = c_p$ for some $c_p > 0$, we have some path $(q_0, f_0) \xrightarrow{w_0} (q_1, f_1) \xrightarrow{w_1} \dots \xrightarrow{w_{k-1}} (q_k, f_k)$ in $\mathcal{G}(\dots)$ with $f_k(q'_p) \geq b - c_p + 1$, $q_i = \pi_i$ for all $0 \leq i \leq k$, and (q_0, f_0) is the initial vertex.
  We need to prove that given some prefix paths $\pi = \pi_0 \dots \pi_{k+1}$ in $\mathcal{R}(\mathcal{A}, w)$ and some prefix path $\pi' = \pi'_0 \dots \pi'_{k+1}$ in $\mathcal{R}(\mathcal{A}', w)$ with $\pi'_{k+1} = q'$ for some $q' \in Q'$, $\mathrm{lag}(\pi, \pi') \leq b$ and $\psi(\pi, \pi') = c$, we have some path $(q_0, f_0) \xrightarrow{w_0} (q_1, f_1) \xrightarrow{w_1} \dots \xrightarrow{w_k} (q_{k+1}, f_{k+1})$ in $\mathcal{G}(\dots)$ with $f_k(q') \geq b - c + 1$, $q_i = \pi_i$ for all $0 \leq i \leq k + 1$, and (q_0, f_0) is the initial vertex.
  - **Case $\pi'_{k+1} \in F'$:**
    * For $\mathrm{lag}(\pi, \pi') \leq b$ to hold, we must have $\mathrm{lag}(\pi_0 \dots \pi_k, \pi'_0 \dots \pi'_k) \leq b$ and $\psi(\pi_0 \dots \pi_k, \pi'_0 \dots \pi'_k) \leq b$. Thus, by the IH, there exists some path $(q_0, f_0) \xrightarrow{w_0} (q_1, f_1) \xrightarrow{w_1} \dots \xrightarrow{w_{k-1}} (q_k, f_k)$ in $\mathcal{G}(\dots)$ with $f_k(\pi'_k) \geq b - \psi(\pi_0 \dots \pi_k, \pi'_0 \dots \pi'_k) + 1$, $q_i = \pi_i$ for all $0 \leq i \leq k$, and $q_k = \pi_k$.
    * For π and π' to be valid, we must further have $(\pi_k, w_k, \pi_{k+1}) \in \delta$ and $(\pi'_k, w_k, \pi'_{k+1}) \in \delta'$.
    * In such a case, the construction of $\mathcal{G}(\dots)$ makes sure that then there is a transition $(q_k, f_k) \xrightarrow{w_k} (q_{k+1}, f_{k+1})$ in $\mathcal{G}(\dots)$ with $q_{k+1} = \pi_{k+1}$ and $f_{k+1}(\pi'_{k+1}) = b + 1$.
    * Since $\pi'_{k+1} \in F'$, we must have $\psi(\pi, \pi') = 0$.
    * Since $\pi'_{k+1} \in F'$ and $\mathrm{lag}(\pi_0 \dots \pi_k, \pi'_0 \dots \pi'_k) \leq b$, we have $\mathrm{lag}(\pi, \pi') \leq b$.
    * Thus, $f_{k+1}(\pi'_{k+1}) = b + 1$ is a correct value for the claim satisfaction.

- **Case** $\pi_{k+1} \notin F \wedge \pi'_{k+1} \notin F'$**:**
  - $\ast$ For $\mathrm{lag}(\pi, \pi') \leq b$ to hold, we must have $\mathrm{lag}(\pi_0 \dots \pi_k, \pi'_0 \dots \pi'_k) \leq b$ and $\psi(\pi_0 \dots \pi_k, \pi'_0 \dots \pi'_k) \leq b$. Thus, by the IH, there exists some path $(q_0, f_0) \xrightarrow{w_0} (q_1, f_1) \xrightarrow{w_1} \dots \xrightarrow{w_{k-1}} (q_k, f_k)$ in $\mathcal{G}(\dots)$ with $f_k(\pi'_k) \geq b - \psi(\pi_0 \dots \pi_k, \pi'_0 \dots \pi'_k) + 1$, $q_i = \pi_i$ for all $0 \leq i \leq k$, and $q_k = \pi_k$.
  - $\ast$ For π and π' to be valid, we must further have $(\pi_k, w_k, \pi_{k+1}) \in \delta$ and $(\pi'_k, w_k, \pi'_{k+1}) \in \delta'$.
  - $\ast$ In such a case, the construction of $\mathcal{G}(\dots)$ makes sure that then there is a transition $(q_k, f_k) \xrightarrow{w_k} (q_{k+1}, f_{k+1})$ in $\mathcal{G}(\dots)$ with $q_{k+1} = \pi_{k+1}$ and $f_{k+1}(\pi'_{k+1}) \geq f_k(\pi'_k)$.
  - $\ast$ Since $\pi_{k+1} \notin F$ and $\pi'_{k+1} \notin F'$, we must have $\psi(\pi, \pi') = \psi(\pi_0 \dots \pi_k, \pi'_0 \dots \pi'_k)$.
  - $\ast$ Since $\pi_{k+1} \notin F$, $\pi'_{k+1} \notin F'$, and $\mathrm{lag}(\pi_0 \dots \pi_k, \pi'_0 \dots \pi'_k) \leq b$, we have $\mathrm{lag}(\pi, \pi') \leq b$.
  - $\ast$ Thus, $f_{k+1}(\pi'_{k+1}) \geq f_k(\pi'_k)$ is a correct value for the claim satisfaction.
- **Case** $\pi_{k+1} \in F \wedge \pi'_{k+1} \notin F'$**:**
  - $\ast$ For $\mathrm{lag}(\pi, \pi') \leq b$ to hold, we must have $\mathrm{lag}(\pi_0 \dots \pi_k, \pi'_0 \dots \pi'_k) \leq b$ and $\psi(\pi_0 \dots \pi_k, \pi'_0 \dots \pi'_k) \leq b$. Thus, by the IH, there exists some path $(q_0, f_0) \xrightarrow{w_0} (q_1, f_1) \xrightarrow{w_1} \dots \xrightarrow{w_{k-1}} (q_k, f_k)$ in $\mathcal{G}(\dots)$ with $f_k(\pi'_k) \geq b - \psi(\pi_0 \dots \pi_k, \pi'_0 \dots \pi'_k) + 1$, $q_i = \pi_i$ for all $0 \leq i \leq k$, and $q_k = \pi_k$.
  - $\ast$ For π and π' to be valid, we must further have $(\pi_k, w_k, \pi_{k+1}) \in \delta$ and $(\pi'_k, w_k, \pi'_{k+1}) \in \delta'$.
  - $\ast$ In such a case, the construction of $\mathcal{G}(\dots)$ makes sure that then there is a transition $(q_k, f_k) \xrightarrow{w_k} (q_{k+1}, f_{k+1})$ in $\mathcal{G}(\dots)$ with $q_{k+1} = \pi_{k+1}$ and $f_{k+1}(\pi'_{k+1}) \geq f_k(\pi'_k) - 1$.
  - $\ast$ Since $\pi'_{k+1} \notin F'$, $\pi_{k+1} \in F$ and $\mathrm{lag}(\pi, \pi') \leq b$, we must have $\psi(\pi, \pi') = \psi(\pi_0 \dots \pi_k, \pi'_0 \dots \pi'_k) - 1$.
  - $\ast$ Since $\pi'_{k+1} \notin F'$, $\pi_{k+1} \in F$, $\psi(\pi_0 \dots \pi_k, \pi'_0 \dots \pi'_k) \leq b - 1$ and $\mathrm{lag}(\pi_0 \dots \pi_k, \pi'_0 \dots \pi'_k) \leq b$, we have $\mathrm{lag}(\pi, \pi') \leq b$.
  - $\ast$ Thus, $f_{k+1}(\pi'_{k+1}) \geq f_k(\pi'_k) - 1$ is a correct value for the claim satisfaction.