Optimizing Büchi automata

Kousha Etessami and Gerard J. Holzmann

Bell Labs, Murray Hill, NJ
{kousha,gerard}@research.bell-labs.com

Abstract. We describe a family of optimizations implemented in a translation from a linear temporal logic to Büchi automata. Such optimized automata can enhance the efficiency of explicit state model checking, as practiced in tools such as Spin.

Some of our optimizations are applied during preprocessing of temporal formulas, others are applied during the translation algorithm itself, while other key optimizations are applied directly to the resulting Büchi automata independent of how they arose. Among these latter optimizations we apply a practical variant of fair simulation reduction based on color refinement. Although there are weaker notions of fair simulation available in the literature, leading to potentially greater reduction, the best algorithms for computing these weaker notions appear at present to be impractical.

We have implemented our optimizations in a translation of an extension to LTL described in [Ete99], which allows expression of all ω-regular properties. Inspired by this work, a subset of the optimizations outlined here has been added to a recent version of Spin. Both implementations begin with an underlying translation based on the algorithm of [GPVW95].

We describe the results of tests we have conducted, both to see how the optimizations improve the sizes of resulting automata, as well as to see how the smaller sizes for the automata affect the running time of Spin's explicit state model checking algorithm. One surprising finding is that, when error paths exist in the model, a smaller number of states in the automaton does not necessarily improve the running time and can actually hurt it in ways that are difficult to predict. Our translation is available via a web-server which includes a GUI that depicts the resulting automata:

`http://cm.bell-labs.com/cm/cs/what/spin/eqltl.html`

1 Introduction

This paper describes a collection of optimization algorithms implemented in a translation from an extension of linear temporal logic to Büchi automata. Büchi automata are finite state automata, viewed not as string acceptors but as acceptors of ω-words. Such ω-automata find wide spread use as modeling and specification mechanisms for concurrent reactive systems, and form the backbone of a family of model checking tools, such as Spin [Hol97], which are based on explicit state enumeration.

In Spin, a linear temporal logic formula φ, used to specify an undesired property of the system[1], is first converted to a Büchi automaton, A_φ. The accepting runs of A_φ represent executions in which the undesirable property holds. This automaton is "producted" with the system model M, in order to determine whether the system has executions

[1] The fragment of LTL used in Spin normally does not allow the "next" operator. This assures that the property specified is stutter-invariant, and hence enables *partial order reduction*, [HP94].

exhibiting this property, as first suggested in [VW86]. The worst case running time of the producting algorithm is $O(|M| \times |A_\varphi|)$. Typically, M is far larger than A_φ, particularly because M itself arises as the product $\Pi_i M_i$ of many state machines M_i describing the concurrent components of the system. Since the size of A_φ is a multiplicative factor in the running time, and yet A_φ is relatively small, it makes sense to put substantial computing effort into minimizing A_φ, before producting takes place. Unfortunately, A_φ is nondeterministic and finding a minimal equivalent nondeterministic automaton is a PSPACE-hard problem[2]. Thus, we can not hope in general to obtain an exact optimal automaton without prohibitive running time. Even a log factor approximation to the optimal size can easily be shown to be PSPACE-hard. We must be content with applying efficient algorithms and heuristics which, in practice, tend to yield small automata.

This paper describes such a collection of optimizations. We have implemented these optimizations atop a translation from an extension of LTL called (SI-)EQLTL studied in [Ete99] which allows the expression of precisely all (*stutter-invariant*) ω-regular languages. Our optimizations do not depend on the details of the temporal logic, beyond ordinary aspects of LTL, so readers only interested in LTL can confine their attention to the LTL fragment. There is a translation from this logic to Büchi automata ([Ete99]), based on the translation of LTL due to Gerth, Peled, Vardi, and Wolper [GPVW95]. That algorithm in practice does not incur the worst-case exponential blow-up necessarily incurred by a naive tableaux construction. However, the algorithm alone still produces automata that are sometimes vastly suboptimal. Thus the need for optimization.

SPIN's LTL translation, also based on [GPVW95], includes some mild forms of optimization. Inspired by this work it was further modified to incorporate some of the rewrite rules discussed here and others listed in [SB00]. We will compare the results of the EQLTL translation with those obtained with SPIN in section 4, both before and after the rewrite rules were added to SPIN.

We partition optimizations into three classes. Figure 1 gives a schematic for the different classes of optimizations. First, before the translation algorithm is applied, the temporal formulas themselves can be simplified and optimized, by applying, e.g., rewrite rules. Secondly, various aspects of the translation algorithm can be "tightened". In this paper we do not discuss any optimizations in this second class. See, e.g., [DGV99]. Third, the resulting automata can be reduced by applying automata theoretic optimizations, including those related to bisimulation and particularly fair simulation reduction.

We assume the reader is familiar with the basic notions of Linear Temporal Logic, and ω-automata. For a general reference on temporal logic the reader is referred to, e.g., [Eme90]. For ω-automata, the reader is referred to [Tho90]. We assume our LTL formulas are defined over a set $P = \{p_1, \ldots, p_n\}$ of propositions, with our word alphabet given by $\Sigma = 2^P$. We use $\varphi \mathrm{V} \psi$ to denote the dual of the until operator, i.e., $\varphi \mathrm{V} \psi = \neg(\neg\varphi \mathrm{U} \neg\psi)$.

In tools such as SPIN, Büchi automata use a more concise labeling notation. Rather than having a distinct transition labeled with each character a of the alphabet $\Sigma = 2^P$, the labels consist of boolean formulas over the atomic propositions of P. Formally, denote

[2] See, e.g., [SM73], from which this result follows, or see also [Koz77].

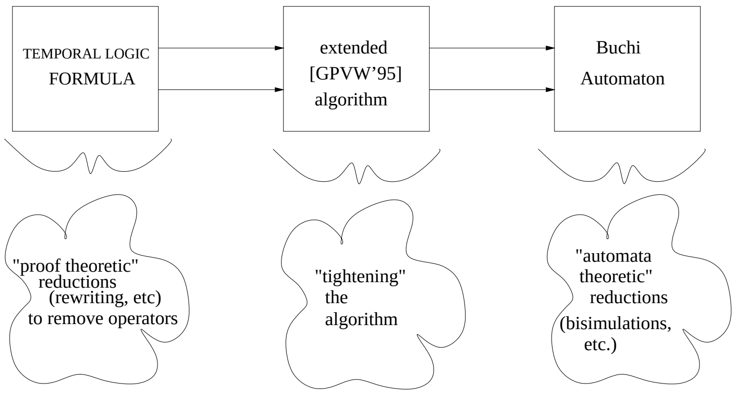

Fig. 1. classes of optimizations

by $\mathcal{B}(P)$, the set of boolean formulas over the atomic propositions P. We assume a Büchi automaton A is given by $\langle Q, \delta, q_0, F\rangle$. Here Q is a set of states, and $\delta \subseteq (Q \times \mathcal{B}(P) \times Q)$ is the transition relation with labels given by the more concise formulas rather than individual characters from $\Sigma = 2^P$. Sometimes, the only formulas we will allow on transitions are *terms*, which are conjunctions of literals. $q_0 \in Q$ is the initial state, and $F \subseteq Q$ is the set of final states. The language $L(A)$ is defined as the set of those ω-words w which have an accepting *run* in A, where a run on w is the sequence of adjacent states that one can visit while traversing the states according to w, and the run is accepting if it infinitely often goes through a state in F.

In translating from LTL to a Büchi automaton (BA), on the way we pass through a *generalized Büchi automaton* (GBA), which rather than one accepting set F, has a family $\mathcal{F}$ of accepting sets, and a run is then said to be accepting if for each $F \in \mathcal{F}$ the run goes infinitely often through a state of F.

In section 2, we describe our proof-theoretic reductions. Section 3 covers our automata-theoretic reductions. In section 4 we discuss our experimental results. We conclude with discussion is section 5.

Note: A preliminary description of this work was presented in July of 1999 at the SPIN workshop on model checking [SPI99]. Independently, F. Somenzi and R. Bloem developed a similar set of optimizations, in a work [SB00] to appear at CAV'2000. Their translation produces generalized Büchi automata rather than ordinary Büchi automata. Their optimizations include a large number of ordinary rewrite rules, but do not include the general rewrite rules we outline using notions of left-append and suffix closure. In addition to simulation reduction, they outline a clever "reverse" simulation reduction, which we do not have. They kindly provided us with their manuscript prior to this submission. Our EQLTL translation was not modified after receipt of their manuscript, however, a version of SPIN's translation (version 3.3.10), tested here against EQLTL, does include some of the rewrite rules from their paper. We have indicated below in which experiments the additional rules were used, and it which they were disabled. In addition to the mentioned web site for the EQLTL translation, as always, the source to the SPIN system is available from the Bell Labs web server for further experimentation.

2 Proof theoretic reductions and rewriting

We begin by describing some simple proof theoretic reductions. These consist of a family of rewrite rules that are applied to formulas recursively, reducing the number of operators and/or connectives. Many such rewrite rules with a similar flavor can be found, e.g., in the text by Manna and Pnueli [MP92]. Since the output size in the translation of [GPVW95] is closely correlated with the number of operators, these reductions can pay off well. From now on, we assume all LTL formulas are in *negation normal form*, meaning negations have been "pushed in" so that they are only applied to atomic propositions.

Definition 1. *A language L of ω-words is said to be left-append closed if for all ω-words $w \in \Sigma^\omega$, and $v \in \Sigma^*$: if $w \in L$, then $vw \in L$.*

The property of left-append closed languages we will exploit in order to reduce the size of our automata is the following:

Proposition 1. *Given an formula ψ_1 such that $L(\psi_1)$ is left-append closed, and given any arbitrary formula γ, the following equivalences hold: (1) $\gamma \mathrm{U} \psi_1 \equiv \psi_1$, (2) $\Diamond \psi_1 \equiv \psi_1$.*

The proof of the first equivalence is straightforward. The second follows from the fact that $\Diamond \psi \equiv \mathtt{true} \mathrm{U} \psi$. Unfortunately, there is no efficient procedure to determine if a property defined by a formula is left-append closed (it is in fact PSPACE-complete). However, there is an easy to check *sufficient* condition for being left-append closed:

Definition 2. *The class of pure eventuality formulas are defined as the smallest set of LTL formulas (in negation normal form) satisfying:*

1. *Any formula of the form $\Diamond \varphi$ is a pure eventuality formula.*
2. *Given pure eventuality formulas ψ_1 and ψ_2, each of $\psi_1 \wedge \psi_2$, $\psi_1 \vee \psi_2$, $\psi_1 \mathrm{U} \psi_2$, $\Box \psi_1$, $\psi_1 \mathrm{V} \psi_2$, and $\bigcirc \psi_1$, is also a pure eventuality formula.*

Lemma 1. *Every pure eventuality formula φ defines a left-append closed property $L(\varphi)$.*

Note: Every LTL definable property that is left-append closed is definable by a pure eventuality formula: If $L(\varphi)$ is left-append closed, then $L(\varphi) = L(\Diamond \varphi)$, and $\Diamond \varphi$ is a pure eventuality formula. However, there might be formulas that are not pure eventuality formulas and yet still define left-append closed properties.

Proof. We show by structural induction on φ that $L(\varphi)$ is left-append closed when φ is a pure eventuality formula. For a word $w = w_0 w_1 \ldots$, let $w[i,j] = w_i w_{i+1} \ldots w_j$ be the substring of w beginning with the i'th character and ending at the j'th. Extending the notation to ω-words, let $w[i,\infty] = w_i w_{i+1} \ldots$ be the ω-word beginning at the i'th character. For the base case, clearly, $\varphi = \Diamond \psi$ is left-append closed, because as long as there is some i such that $w[i,\infty] \models \psi$ we can arbitrarily delay that point with an added finite prefix v without violating $\Diamond \psi$. Inductively, suppose formulas ψ_i are pure eventualities and φ has the form:

1. $\underline{\varphi = \psi_1 \wedge \psi_2}$. If $w \models \psi_1 \wedge \psi_2$, then $w \models \psi_1$ and $w \models \psi_2$, and thus, since ψ_1 and ψ_2 are left-append closed, $vw \models \psi_1$ and $vw \models \psi_2$, and thus $vw \models \psi_1 \wedge \psi_2$.
2. $\underline{\varphi = \psi_1 \vee \psi_2}$. This case is very similar to the previous one.
3. $\underline{\varphi = \psi_1 \mathrm{U} \psi_2}$. Suppose $w \models \psi_1 \mathrm{U} \psi_2$. Thus there is some $i \geq 0$ such that $w[i, \infty] \models \gamma$, and for all $j < i$, $w[j, \infty] \models \psi_1$. By induction ψ_2 defines a left-append closed property, so, for any $v \in \Sigma^*$ such that $|v| = k$, we have that $vw[0, \infty] \models \psi_2$. Hence $vw[0, \infty] \models \psi_1 U \psi_2$. Thus φ is left-append closed.
4. $\underline{\varphi = \Box \psi_1}$. Suppose $w \models \Box \psi_1$. Then in particular, $w \models \psi_1$, and since ψ_1 is left-append closed, for every suffix $v'w$ of vw, $v'w \models \psi_1$, and hence $vw \models \Box \psi_1$.
5. $\underline{\varphi = \psi_1 \mathrm{V} \psi_2}$. Suppose $w \models \psi_1 \mathrm{V} \psi_2$. Now $\psi_1 \mathrm{V} \psi_2$ is equivalent to $(\Box \psi_2) \vee (\psi_2 \mathrm{U} (\psi_1 \wedge \psi_2))$. We have already shown in the proofs above that left-append closed formulas remain left-append closes under the application of $\vee$, $\wedge$, and U, and thus by induction the given formula $\psi_1 \mathrm{V} \psi_2$ also remains left-append closed.
6. $\underline{\varphi = \bigcirc \psi_1}$. Suppose $w \models \bigcirc \psi_1$. Then, since ψ_1 is left-append closed, $w \models \psi_1$. Now let $v = av'$, for any $a \in \Sigma$. since ψ_1 is left-append closed, $v'w \models \psi_1$, and thus $vw \models \bigcirc \psi_1$. Hence $\varphi = \bigcirc \psi_1$ is left-append closed.

□

Just as we defined left-append closed properties and pure eventuality formulas, we can also consider suffix closed properties and pure universality formulas (formulas are always in negation normal form):

Definition 3. *A language L is* suffix closed *if whenever $w \in L$ and w' is a suffix of w, then $w' \in L$.*

Proposition 2. *For a formula ψ with a suffix closed language $L(\psi)$, and an arbitrary formula γ, the following equivalences hold: (1) $\gamma \mathrm{V} \psi \equiv \psi$, (2) $\Box \psi \equiv \psi$.*

We have the following sufficient condition for a formula to define a suffix closed language.

Definition 4. *The class of* purely universal *formulas is defined inductively as the smallest set of formulas satisfying:*

1. *Any formula of the form $\Box \varphi$ is purely universal.*
2. *Given purely universal formulas ψ_1 and ψ_2, any formula of the form: $\psi_1 \wedge \psi_2$, $\psi_1 \vee \psi_2$, $\psi_1 \mathrm{V} \psi_2$, $\Diamond \psi_1$, $\psi_1 \mathrm{U} \psi_2$, and $\bigcirc \psi_1$ is also purely universal.*

Lemma 2. *Every pure universality formula defines a suffix closed property.*

From lemmas 1 and 2, and propositions 1 and 2, the following lemma follows, describing our basic rewrite rules:

Lemma 3. (Basic Operator Reduction Lemma) *For all LTL formulas φ, ψ, and γ, the following equivalences hold:*

1. $(\varphi \mathrm{U} \psi) \wedge (\gamma \mathrm{U} \psi) \equiv (\varphi \wedge \gamma) \mathrm{U} \psi$
2. $(\varphi \mathrm{U} \psi) \vee (\varphi \mathrm{U} \delta) \equiv \varphi \mathrm{U} (\psi \vee \delta)$

3. $\Diamond(\varphi \mathrm{U} \psi) \equiv \Diamond\psi$
4. Whenever ψ *is a pure eventuality formula* $(\varphi \mathrm{U} \psi) \equiv \psi$*, and* $\Diamond\psi \equiv \psi$*.*
5. Whenever ψ *is a pure universality formula* $(\varphi \mathrm{V} \psi) \equiv \psi$*, and* $\Box\psi \equiv \psi$*.*

Note that in each of the equivalences above the right hand side has at least one fewer temporal operator than the left hand sides. A formula (or subformula) that fits the pattern on the left hand side is replaced by the one on the right. The first three equivalences each have corresponding duals which are also applied.

There are, as you can imagine, many other rules one could use (see, e.g., [MP92]). Our aim has not been to list exhaustively all such rules, but to list a few that have direct impact without excessive cost.

3 Automata Theoretic reductions

In this section we describe the reduction techniques which are applied directly to the Büchi automata produced by the algorithm of the previous section. The main algorithm in this section is a variant of "fair" simulation reduction, which itself is a natural generalization of bisimulation reduction. Before that algorithm, however, we first describe some other reductions. The reductions covered in this entire section are *complementary*, in the sense that applying one reduction can subsequently enable further gain from another reduction, until a "fixed-point" is reached where we can gain no more.

Simplifying edge terms The concise notation for automata, namely term (or formula) labeled transitions, gives us an opportunity to perform some optimizations to reduce the number of edges further, by combining terms, or more generally, reducing formulas. For example, whenever we encounter two transitions: $(q_1, (p_1 \wedge p_2), q_2) \in \delta$ and $(q_1, (p_1 \wedge \neg p_2), q_2) \in \delta$, we can combine the two into one simplified transition: $(q_1, p_1, q_2) \in \delta$, using the obvious propositional rule: $(p_1 \wedge p_2) \vee (p_1 \wedge \neg p_2) \leftrightarrow p_1$. Again, we can perform a vast family of such reductions, this time based on propositional logic proof rules rather than temporal proof rules. We will not elaborate further on these optimizations since they are a standard part of elementary propositional logic. See, e.g., [End72].

Removing "never accepting" nodes This is a trivial optimization, but nevertheless quite important in the context of the other optimizations. In a Büchi automaton A, a node q that does not have any accepting run leading from it can safely be removed without changing $L(A)$. We compute the set of states with a run leading infinitely often to an accepting state by computing the strongly connected components of A, and retaining those states that lead to non-trivial SCCs containing an accept state.

Removing fixed-formula balls This is an interesting reduction, which is not correct for ordinary *finite automata*, but is correct for Büchi automata. We assume we have already removed "never-accepting" nodes. The basic idea of the reduction is that, if in a

Büchi automaton we are ever stuck in a component from which we can not get out, and the only transition labels in this component are α, and there is some accepting state in this component, then we can treat this entire component as a single accepting state with a self-transition labeled by α. More formally:

Definition 5. *For $\alpha \in \mathcal{B}(P)$, a* **fixed-formula α-ball** *inside a Büchi automaton A is a set $Q' \subseteq Q$ of nodes such that*

1. *α is the unique formula which labels the transitions inside Q', i.e., if $q'_1, q'_2 \in Q'$ and $\delta(q'_1, \xi, q'_2) \in \delta$, then $\xi = \alpha$.*
2. *The nodes Q' form a strongly connected component of the underlying graph G of A, where $V_G = Q$ and $E_G(q_1, q_2) \iff \exists \sigma\ (q_1, \sigma, q_2) \in \delta$.*
3. *There is no transition leaving Q', i.e., no $(q', \xi, q) \in \delta$, where $q' \in Q'$ and $q \notin Q'$.*
4. *Q' contains an accepting state, i.e., $Q' \cap F \neq \emptyset$*

Proposition 3. *Given a Büchi automaton $A = \langle Q, \delta, q_0, F\rangle$, suppose $Q' \subseteq Q$ is a fixed-formula α-ball of A. Let $A' = \langle (Q \setminus Q') \cup \{q_{new}\}, \delta', q'_0, (F \setminus Q') \cup \{q_{new}\}\rangle$, where the transitions involving q_{new} are $(q_1, \xi, q_{new}) \in \delta'$ whenever there was some $q' \in Q'$ such that $(q_1, \xi, q') \in \delta$, and $(q_{new}, \alpha, q_{new}) \in \delta'$, and all transitions inside $Q \setminus Q'$ remain the same. Moreover, $q'_0 = q_{new}$ if $q_0 \in Q'$, and otherwise $q'_0 = q_0$. Then $L(A) = L(A')$.*

Although this looks like a rather specialized reduction, fixed-formula balls are frequently introduced into Büchi automata as a result of the translation from a GBA, A', to a BA, A. Consider, for example, the simple situation where there is an accepting state q with the unique transition $(q, \texttt{true}, q) \in \delta'$ leaving q in A'. Then, if there are k generalized Büchi accepting sets in A', there will be k copies of q, call them $q_0, \ldots, q_{k-1}$, in A forming a k-state loop with $(q_i, \texttt{true}, q_{(i+1 \bmod k)}) \in \delta$. This is a fixed-formula ball which can be collapsed to one state and one transition, potentially enabling further reductions.

3.1 Reductions based on bisimulation and simulation

In this section we describe what, algorithmically, is our most elaborate reduction: a kind of fair simulation reduction, with an algorithm based on color refinement (see, e.g., [HU79]). It turns out that there are several distinct notions of fair simulation in the literature (see, e.g., [HKR97]). The "weaker" the notion, the more reduction it potentially enables. The notion we chose to implement is by no means the weakest available notion, but its advantage is that it admits an easy to implement and reasonably efficient algorithm which is a very natural modification of the ordinary color refinement algorithm. Henzinger, Kupferman, and Rajamani, [HKR97], describe a weaker notion of fair simulation, and give a polynomial time algorithm for computing the relation. But their algorithm, which employs an algorithm of [HHK95] for computing a maximal simulation relation and then resorts to tree automata and their emptiness problem to deal with fair simulation, is apparently impractical, requiring a worst case running time of $O(n^6)$.[3]

[3] There is a typographical error in the conference version of [HKR97] which would seem to indicate a running time of $O(n^4)$, but this typo has been corrected in more recent versions.

Review: basic partition refinement We first review the standard bisimulation reduction algorithm (see, e.g.,[KS90]), based on color-refinement partitioning of the states, accounting now for the fact that edges are labeled by terms rather than individual characters of the alphabet. The fact that labels are terms rather than arbitrary formulas helps us when we switch to simulation reduction.

The basic algorithm is given in Figure 2. On input $A = \langle Q, \delta, q_0, F\rangle$, the algorithm proceeds as follows to create an output A', such that $L(A) = L(A')$. It creates a coloring function $C^i : Q \mapsto \{1, \ldots, |Q|\}$ which initially incorporates the acceptance condition by assigning one color to accept states and a different color to reject states. It is refined after each iteration i until a fixed point is reached, namely, no new colors are created. We use $C^i(Q)$ to denote the set of colors after round i. Although during the algorithm our notation assigns *tuples* as colors, these tuples can easily converted to numbers again after each iteration, by the usual lexicographic sorting and renaming.

proc *BasicBisimReduction*(A) $\equiv$
  /* Initialize: $\forall q \in Q$ $C^{-1}(q) := 1$, and $\forall q \in F$ $C^0(q) := 1$, $\forall q \in Q \setminus F$ $C^0(q) := 2$. */
  $i := 0$;
  while $|C^i(Q)| \neq |C^{i-1}(Q)|$ **do**
    $i := i + 1$;
    foreach $q \in Q$ **do**
      $C^i(q) := \langle C^{i-1}(q), \cup_{(q,\tau,q')\in\delta}\{(C^{i-1}(q'), \tau)\}\rangle$
    od
    Rename color set $C^i(Q)$, with $\{1, \ldots, |C^i(Q)|\}$, using lexicographic ordering.
  od
  $C := C^i$; **return** $A' := \langle Q' := C(Q), \delta', q'_0 := C(q_0), F' := C(F)\rangle$;
  /* δ' defined so that $(C(q_1), \tau, C(q_2)) \in \delta'$ */
  /* if and only if $(q_1, \tau, q_2) \in \delta$ for $q_1, q_2 \in Q$ */

Fig. 2. Basic Bisimulation Reduction Algorithm

This algorithm is correct for nondeterministic finite automata as well as Büchi automata. For NFAs this is because the following inductive invariant is maintained by the algorithm. Let $S_i^q = \{s \in \cup_{j\leq i}\Sigma^j \mid q \overset{s}{\rightsquigarrow} q''\ , q'' \in F\}$ be the set of strings of length at most i with which one can reach an accept state from state q.

Proposition 4. *For all* $q, q' \in Q$, *if* $C^i(q) = C^i(q')$ *then* $S_i^q = S_i^{q'}$.

After at most $|Q|$ iterations, the color refinement reaches a fixed point, and thus if $C(q) = C(q')$ at this point, then $\forall l \geq 0$, $S_l^q = S_l^{q'}$. For ω-automata, in particular for Büchi automata, the proof is only slightly more subtle.

Theorem 1. *(see, e.g., [KS90]) Given a Büchi automaton A, the* BasicBisimReduction *algorithm produces a Büchi automaton A' such that $L(A) = L(A')$.*

Modifying the refinement algorithm, simulation: We first illustrate the weakness of the ordinary color refinement. Consider the automaton in Figure 3. From the initial state q_0, reading p_1 in the first character of the input, w_0, we can go directly to the accepting state q_1, in which we can stay regardless of the rest of the ω-word, $w_1 w_2 \ldots$. Therefore, the transition from q_0 to q_2 labeled by the term $p_1 \wedge p_2$ is completely redundant. Thus, in fact we can eliminate both the state q_2, and all the subsequent states reachable from q_2 without affecting the language of the automaton. The trouble with ordinary bisimulation reduction is that it doesn't recognize this situation, nor the more complicated situations like it where one transition from a state "subsumes" another.

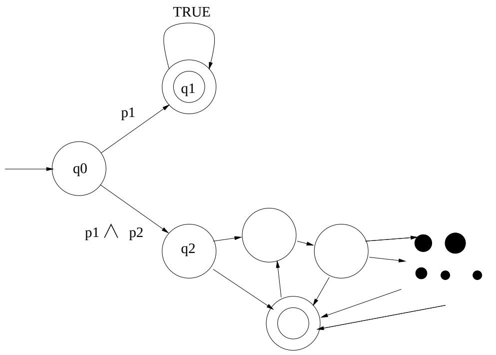

Fig. 3. Why simulation based reduction is better than bisimulation based reduction

Using simulation reduction instead of bisimulation, immediately remedies this weakness. Algorithmically for us, this corresponds to not just partitioning the states into equivalence classes, but maintaining a quasi-order on the states (a reflexive, transitive relation), which essentially defines which classes of states "subsume" others.

Our algorithm amounts to computing a strong version of a fair simulation relation, one that works for both finite and ω-automata.(This kind of fair simulation is termed *direct* simulation and used in the independent work of [SB00], and as they point out, had been previously used in [DHWT91].) As pointed out before, algorithms with better complexity ([HHK95]), as well as algorithms that yield greater reduction but have worst complexity ([HKR97]) exist in the recent literature. We chose our implementation based on its simplicity and relatively good performance in practice.

The algorithm we implement is a natural revision of the basic partition algorithm in Figure 2, but rather than refining a partition of the vertices, inductively refines a quasi-order on the vertices. Instead of maintaining the entire quasi-order, we can keep a partial order, $po^i_{\leq}(x, y)$, on the equivalence (color) classes that exist in the quasi-order after round i. We associate with the neighbors q' of each node q, such that for some σ, $(q, \sigma, q') \in \delta$, a **neighbor i-type** $(C^i(q'), \sigma)$. Consider the i-types $(C^i(q''), \tau)$ and $(C^i(q'), \sigma)$ where τ and σ are terms labeling transitions. We say that $(C^i(q'), \sigma)$ *i-dominates* $(C^i(q''), \tau)$ if $po^i_{\leq}(C^i(q''), C^i(q'))$ and σ is a subterm of τ (i.e., as a boolean formula τ implies σ). For a node q, and an edge $(q, \tau, q') \in \delta$, we say that the pair $(C^i(q'), \tau)$ is *i-maximal* for q, if there is no q'' with $(q, \sigma, q'') \in \delta$, such that $(C^i(q''), \sigma)$

i-dominates $(C^i(q'), \tau)$. Given $q \in Q$, let the set of i-maximal neighbor i-types of q be given by $\mathbf{N^i(q)} = \{(C^i(q'), \tau) \mid (q, \tau, q') \in \delta$ and $(C^i(q'), \tau)$ is i-*maximal*$\}$. We will say that $N^i(q')$ i-*dominates* $N^i(q)$, if for every $(c, \tau) \in N^i(q)$ there is a pair $(c', \sigma) \in N^i(q')$ such that (c', σ) i-dominates (c, τ).

proc *StrongFairSimulationReduction*(A) $\equiv$
  /* Initialize: $\forall q \in Q$ $C^{-1}(q) := 1$, and $\forall q \in F$ $C^0(q) := 1$, $\forall q \in Q \setminus F$ $C^0(q) := 2$. */
  /* Initialize the partial order on colors: */
  $po^0_\leq(2,1) := true$; $po^0_\leq(1,1) := true$; $po^0_\leq(2,2) := true$; $po^0_\leq(1,2) := false$;
  $i := 0$;
  while $|C^i(Q)| \neq |C^{i-1}(Q)|$ or $|po^i| \neq |po^{i-1}|$ **do**
      $i := i + 1$;
      foreach $q \in Q$ **do** $C^i(q) := \langle C^{i-1}(q), N^{i-1}(q)\rangle$ **od**
      /* now we update the partial order, creating $po^i_\leq$ */
      foreach $(c^i_1 = \langle c^{i-1}_1, N^{i-1}_1\rangle) \in C^i(Q)$ **do**
        foreach $(c^i_2 = \langle c^{i-1}_2, N^{i-1}_2\rangle) \in C^i(Q)$ **do**
          if $po^{i-1}_\leq(c^{i-1}_2, c^{i-1}_1)$ and N^{i-1}_1 $(i-1)$-*dominates* N^{i-1}_2
            then $po^i_\leq(c^i_2, c^i_1) :=$ `true`;
            else $po^i_\leq(c^i_2, c^i_1) :=$ `false`;
          fi
        od
      od
      Rename color set $C^i(Q)$, with $\{1, \ldots, |C^i(Q)|\}$, using lexicographic ordering.
      Adapt $po^i_\leq$ to these renamed colors.
  od
  $C := C^i$; **return** $A' := \langle Q' := C(Q), \delta', q'_0 := C(q_0), F' := C(F)\rangle$;
  /* δ' defined so that $(C(q_1), \tau, C(q_2)) \in \delta'$ */
  /* if and only if $(C(q_2), \tau) \in N^i(q_1)$ */.

Fig. 4. Improved reduction algorithm based on strong fair simulation

Now, consider the algorithm in Figure 4. The algorithm first initializes the coloring and the partial order, so that accept nodes get a "greater" color than reject nodes. It then iteratively refines this partition, using the fact that, in the interim, when one neighbor is dominated by another, only the dominating neighbor needs to be considered in the next round of coloring. The partial order itself is upgraded using the fact that, if the neighbors of color class c dominate the neighbors of class c', and the "old color" of class c is greater than the "old color" of c', then in the new coloring, c dominates c'.

The algorithm halts when, neither the number of colors nor the partial order on the colors changes from one iteration to the next. There is however, a trick used to speed up this check: rather than checking that the entire partial order remains the same, all we need to do is check that the number of pairs, $|po^i|$, in the partial order do not change. This is so because the effect of the loop on the underlying quasi-order is monotone, meaning edges are only being removed from the quasi-order during this fixed-point computation and never added. The correctness of the algorithm for finite and ω-automata, respectively are the following two claims (we omit proofs):

Proposition 5. *If $C^i(q) \leq C^i(q')$ then $S^i_q \subseteq S^i_{q'}$*

Theorem 2. *Given a Büchi or finite automaton A, the StrongFairSimulationReduction algorithm constructs A', such that $L(A) = L(A')$.*

Complexity: An upper bound for running time of the algorithm can be obtained as follows: the main while loop dominates the running time. It can iterate at most m times, where m is the number of transitions in the original automaton. The body of the loop requires $O(n + k^2c)$ time, where n is the number of states in the input automaton, k is the number of states in the resulting output automaton, and c is the time required to compute the i-dominates relation on neighbor sets. Thus, the total worst case running time is bounded by $O(mk^2c + mn)$.

This is by no means the best possible: [HHK95] show that a maximal simulation relation can be computed in time $O(mn)$, and the same algorithm can be modified to accommodate acceptance conditions as in our setting. The main benefit of our algorithm is that it is easy to implement. Also, it rarely requires the worst case time: often the while loop will iterate far fewer than m times. In practice, we find this algorithm runs fairly quickly.

4 Experimental Results

We now describe a number of experiments we have conducted on three different implemented translations to Büchi automata:

1. **GPWV95:** The first translation is based solely on the algorithm of [GPVW95], implemented in ML by Doron Peled. To be compatible, the resulting GBA from this translation is converted to a BA using a straightforward construction, and the labels are moved from states to transitions.
2. SPIN **version 3.3.10:** SPIN's LTL translation is based on [GPVW95], with a number of relatively simple additional optimizations in the third phase of the algorithm. They consist primarily of merging identical states, and removing simple cases of two-node *balls*. This version of SPIN uses some additional rewrite rules from the current paper and some taken from [SB00]. For comparison, we also include in Table 1 results, marked SPIN-, for the same version of the tool with all new rewrite rules and optimizations *disabled*.
3. **EQLTL:** This is the algorithm which includes all the optimizations we have mentioned in this paper, implemented in ML on top of the [GPVW95] algorithm. Although the logic used in this algorithm, (SI-)EQLTL ([Ete99]), allows us to express strictly more properties than LTL, namely all (stutter-invariant) ω-regular properties, we confine our experiments to the standard LTL fragment in order to be able to compare our results to the other two translations. Table 1 includes measurements for 4 versions of EQLTL with specific classes of optimization enabled or disabled. The version titled "EQLTL" performs all reduction, "EQLTL-auto" performs only the

automata theoretic reductions, "EQLTL-rewr" only the rewrite rules, and "EQLTL-none" performs only bare-bones trivial reductions, like removing dead states after converting from a GBA to a BA.

We performed two sets of measurements. In the first set we randomly generated 500 LTL formulas with up to 10 temporal operators and/or boolean connectives, and 3 propositions. This set is generated by randomly choosing grammar rules from LTL's context-free grammar, expanding a formula up to its maximum length. Similar random formula generation was also used in [DGV99], [Tau99], and [SB00]. For comparison, the second set includes 12 hand selected formulas, shown below Figure 5, including many that are in common use.

	Random formulas		12 formulas		version
	states	*average*	*states*	*average*	
GPVW	13400	26.8	183	15.25	basic algorithm
SPIN-	4069	8.1	77	6.4	minus rewrite rules
SPIN	3419	6.8	72	6	with rewrite rules
EQLTL-none	6473	12.9	117	9.8	no non-trivial optimizations
EQLTL-rewr	5599	11.2	114	9.5	rewrite rules only
EQLTL-auto	2709	5.4	51	4.3	automata optimizations only
EQLTL	2564	5.1	49	4.1	all optimizations
	transitions	*average*	*transitions*	*average*	
GPVW	42185	84.37	2321	193.4	basic algorithm
SPIN-	13982	27.9	161	13.4	minus rewrite rules
SPIN	11245	22.5	156	13	with rewrite rules
EQLTL-none	18511	37.0	232	19.3	no non-trivial optimizations
EQLTL-rewr	14774	29.5	228	19.0	rewrite rules only
EQLTL-auto	4952	11.0	86	7.2	automata optimizations only
EQLTL	4952	9.9	81	6.75	all optimizations

Table 1. 500 random and 12 specific formulas

Table 1 shows the total number of states and transitions generated by the three main translations for the 500 random formulas, and the averages for each. The graph on the upper left in Figure 5 shows, along the x axis the 500 formulas, sorted by the number of states in the output of the EQLTL algorithm for each formula, and on the y axis the number of states for each formula for EQLTL. The graph on the upper right in this figure shows the formulas in the same order, but with on the y axis the difference between the numbers of states in the automata generated by the most recent version of SPIN and by EQLTL. A value of zero means that the two automata have the same number of states. Negative numbers mean that the EQLTL automaton is smaller than the SPIN automaton by the amount shown. In most cases the automata are fairly close in size, but there are notable exceptions, predominantly in favor of the EQLTL algorithm.

The graph in the lower left of Figure 5 shows the number of states that is generated by the three algorithms for the 12 hand selected LTL formulas from our second test set. Three bars are shown for each formula, the first give the number of states generated by

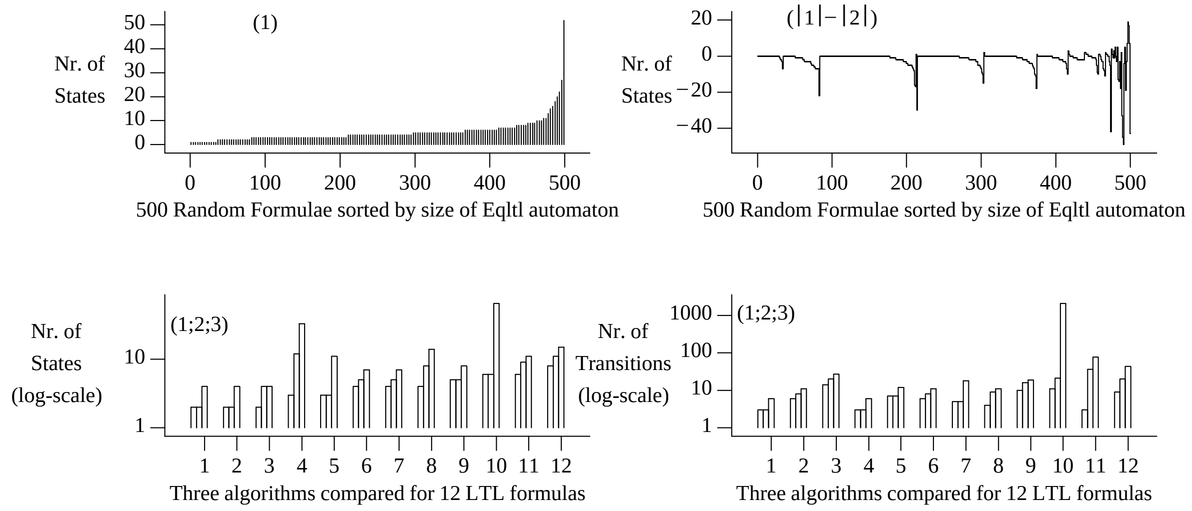

Fig. 5. — Automata Sizes (1=Eqltl, 2=Spin, 3=Gpvw).

1. $p \; U \; (q \wedge \Box r)$
2. $p \; U \; (q \wedge X(r \; U \; s))$
3. $p \; U \; (q \wedge X(r \wedge (\Diamond (s \wedge X(\Diamond (t \wedge X(\Diamond (u \wedge X\Diamond v))))))))$
4. $\Diamond (p \wedge X \Box q)$
5. $\Diamond (p \wedge X(q \wedge X(\Diamond r)))$
6. $\Diamond (q \wedge X(p \; U \; r))$
7. $(\Diamond \Box q) \vee (\Diamond \Box p)$
8. $\Box (p \rightarrow (q \; U \; r))$
9. $\Diamond (p \wedge X\Diamond (q \wedge X\Diamond (r \wedge X\Diamond s)))$
10. $\Box\Diamond p \wedge \Box\Diamond q \wedge \Box\Diamond r \wedge \Box\Diamond s \wedge \Box\Diamond t$
11. $(p \; U \; q \; U \; r) \vee (q \; U \; r \; U \; p) \vee (r \; U \; p \; U \; q)$
12. $\Box (p \rightarrow (q \; U \; (\Box r \vee \Box s)))$

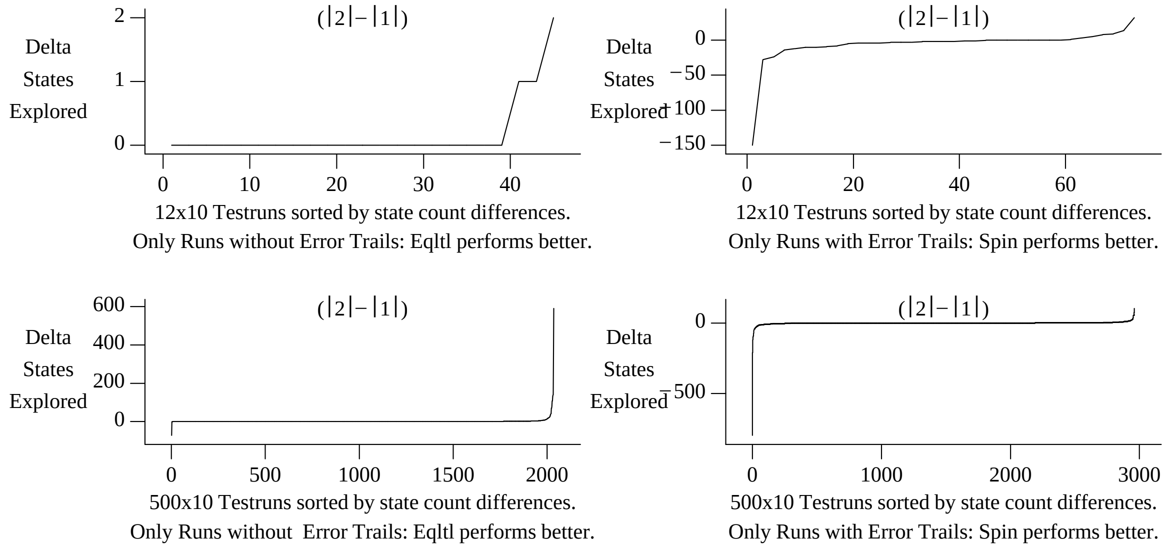

Fig. 6. — Model Checking Performance (1=Eqltl, 2=Spin).

EQLTL, the second by SPIN, and the third by GPVW. The graph in the lower right of the figure shows the same data for the number of transitions in each automaton. In both cases the y axis is plotted on a logarithmic scale.

We have also measured how the sizes of the automata affect the running time of SPIN's model checking algorithm, to test the hypothesis that reducing automata size should help reduce expected running time of the model checker. Each formula was tested against randomly generated systems with 50 states each. Propositional values were changed randomly along the edges in each of these systems, similar to [Tau99]. A standard model checking run was performed for each formula from our test set against 10 such randomly generated systems. The model checking runs halted when the formula was either shown to be satisfied or not satisfied for the system. Figure 6 compares the number of combined states that were explored in the product of the automaton with the system for each model checking run. The graphs show the *difference* between the number of states explored for the EQLTL automaton and the number of states explored by SPIN's automaton on the y axis. A negative number (on the left side of the graphs) represents the cases where more states were explored for the EQLTL automaton than for SPIN's automaton. Positive numbers (on the right side of the graphs) represent cases where the EQLTL automaton allowed the model checker to explore fewer states.

The graphs illustrate that for system models without violating runs the number of states explored, as expected, is larger when the size of the property automaton is larger. However, somewhat surprisingly, when there are violating runs, the benefit of smaller property automata is not clear. In particular, although the EQLTL translation produces smaller automata, in most cases it explores as many states as SPIN, and in a few cases it explores more. We have no adequate explanation for this phenomenon. Since the SPIN automata on average have about twice the number of transitions of the EQLTL automata, it is possible that for a given number of states an increase in the number of transitions can contribute positively to model checking performance, but this needs to be explored.

The running times for all the EQLTL optimizations are in most cases faster than SPIN's and GPVW's.

5 Conclusions

The realm of possible heuristic optimizations for Büchi automata is vast: the PSPACE-hardness of finding the optimal (or even approximately optimal) sized automaton leaves an opening for any number of optimizations. We have outlined those optimizations we have found most useful. There is, however, plenty of room for improvement, and we anticipate adding incremental improvements to our translation in the future.

The optimized (SI-)EQLTL translation is available, through a GUI that draws the resulting automaton and provides the corresponding SPIN code via a web server at `http://cm.bell-labs.com/cm/cs/what/spin/eqltl.html`. The implementation is in the language ML, based on an implementation of the algorithm of [GPVW95] by Doron Peled. SPIN's LTL translation is written in C.

Acknowledgement. Thanks to Doron Peled for providing us his original ML code for the [GPVW95] algorithm, to Fabio Somenzi and Roderic Bloem for providing us their unpublished manuscript [SB00], and to the anonymous reviewers for helpful comments. Thanks to Steven Eker for pointing out an error.

References

[DGV99] M. Daniele, F. Giunchiglia, and M. Y. Vardi. Improved automata generation for linear temporal logic. In *Proc. of 11th Int. Conf. on Computer Aided Verification (CAV99)*, number 1633 in LNCS, pages 249–260, 1999.

[DHWT91] D. L. Dill, A. J. Hu, and H. Wong-Toi. Checking for language inclusion using simulation relations. In *Proceedings of CAV'91*, pages 329–341, 1991.

[Eme90] E. A. Emerson. Temporal and modal logics. In J. van Leeuwen, editor, *Handbook of Theoret. Comput. Sci.*, volume B, pages 995–1072. Elsevier, Amsterdam, 1990.

[End72] H. B. Enderton. *A Mathematical Introduction to Logic.* Academic Press, New York, 1972.

[Ete99] K. Etessami. Stutter-invariant languages, ω-automata, and temporal logic. In *Proceedings of the 11th International Conference on Computer Aided Verification*, pages 236–248, 1999.

[GPVW95] R. Gerth, D. Peled, M. Y. Vardi, and P. Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *Protocol Specification Testing and Verification*, pages 3–18, 1995.

[HHK95] M. Henzinger, T. Henzinger, and P. Kopke. Computing simulations on finite and infinite graphs. In *Proc. of 36th IEEE Symp. on Foundations of Comp. Sci. (FOCS'95)*, pages 453–462, 1995.

[HKR97] T. Henzinger, O. Kupferman, and S. Rajamani. Fair simulation. In *Proc. of 9th Int. Conf. on Concurrency Theory (CONCUR'97)*, number 1243 in LNCS, pages 273–287, 1997.

[Hol97] G. J. Holzmann. The model checker spin. *IEEE Transactions on Software Engineering*, 23(5):279–295, 1997.

[HP94] G. J. Holzmann and D. Peled. An improvement in formal verification. In *7th International Conference on Formal Description Techniques*, pages 177–194, 1994.

[HU79] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Formal Languages and Computation.* Addison-Wesley, 1979.

[Koz77] D. Kozen. Lower bounds for natural proof systems. In *18th IEEE Symposium on Foundations of Computer Science*, pages 254–266, 1977.

[KS90] P. C. Kanellakis and S. A. Smolka. CCS expressions, finite state processes, and three problems of equivalence. *Information and Computation*, 86(1):43–68, 1990.

[MP92] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems*, volume 1. Springer-Verlag, Berlin/New York, 1992.

[SB00] F. Somenzi and R. Bloem. Efficient büchi automata from ltl formulae. In *Proceedings of 12th Int. Conf. on Computer Aided Verification*, 2000.

[SM73] L. Stockmeyer and A. R. Meyer. Word problems requiring exponential time: preliminary report. In *Proceedings of 5th ACM Symposium on Theory of Computing*, pages 1–9, 1973.

[SPI99] *SPIN workshop on theoretical aspects of model checking*, July 1999. Trento, Italy.

[Tau99] H. Tauriainen. A randomized testbench for algorithms translating linear temporal logic formulae into büchi automata. In *Proc. Workshop Concurrency, Specifications, and Programming*, pages 251–262, Warsaw, Poland, Sept. 1999.

[Tho90] W. Thomas. *Handbook of Theoretical Computer Science, volume B*, chapter Automata on Infinite Objects, pages 133–191. MIT Press, 1990.

[VW86] M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In Dexter Kozen, editor, *First Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 322–331, 1986.