Efficient Büchi Automata from LTL Formulae*

Fabio Somenzi and Roderick Bloem

Department of Electrical and Computer Engineering
University of Colorado, Boulder, CO, 80309-0425
{Fabio,Roderick.Bloem}@Colorado.EDU

Abstract. We present an algorithm to generate small Büchi automata for LTL formulae. We describe a heuristic approach consisting of three phases: rewriting of the formula, an optimized translation procedure, and simplification of the resulting automaton. We present a translation procedure that is optimal within a certain class of translation procedures. The simplification algorithm can be used for Büchi automata in general. It reduces the number of states and transitions, as well as the number and size of the accepting sets—possibly reducing the strength of the resulting automaton. This leads to more efficient model checking of linear-time logic formulae. We compare our method to previous work, and show that it is significantly more efficient for both random formulae, and formulae in common use and from the literature.

1 Introduction

The standard approach to LTL model checking [18, 14] consists of translating the negation of a given LTL formula into a Büchi automaton, and checking the product of the property automaton and the model for language emptiness. The quality of the translation affects the resources required by the model checking experiment. This motivates the search for algorithms that generate efficient automata, i.e., automata with few states, few transitions, and simple acceptance conditions.

The initial approaches to translation of LTL formulae were not designed to yield small automata. The process of [18] always yields the worst-case result of $O(2^n)$ states, where n is the length of the formula. The approach of [11] was the first to produce automata that were not necessarily of worst-case size. In [9], a more efficient algorithm was proposed that works on-the-fly. A further improvement over this algorithm, based on syntactic simplification, was discussed in [5].

We present an approach to the generation of Büchi automata from LTL formulae that extends the work of [9, 5] in three ways:

1. It applies rewriting rules to the formula before translation.
2. It reduces the number of states generated by the translation by applying boolean optimization techniques.
3. It simplifies both the transition structure and the acceptance conditions of the resulting Büchi automaton.

* This work was supported in part by SRC contract 98-DJ-620 and NSF grant CCR-99-71195.

The last phase of our algorithm is independent of LTL, and can be applied whenever one is interested in simplifying a Büchi automaton. Our algorithm is not designed to work on-the-fly, since the automata generated by the algorithm are typically much smaller than the model.

Both explicit and implicit model checking algorithms benefit from the simplification of the Büchi automata. Explicit techniques check emptiness of the product automata in time proportional to the size of the property automaton. Symbolic algorithms need, in general, a number of symbolic operations that is quadratic in the size of the property automaton [4, 12]. We can hence expect appreciable speedups in emptiness checks using either technique if we can reduce the size of the automaton significantly.

The *strength* of a Büchi automaton [13, 2] relates to the complexity of the procedure required to symbolically model check the corresponding property. For a *strong* automaton, an emptiness check requires the computation of a μ-calculus formula of alternation depth 2, which takes a number of preimage computations quadratic in the size of the automaton. A *weak* automaton requires an alternation-free greatest fixpoint computation, and hence only linearly many preimage computations. Finally, a *terminal* automaton only requires reachability analysis, and is therefore amenable to on-the-fly model checking. Our procedure tends to produce results of lesser strength than results of previous algorithms.

We compare our technique to those of [9] and [5]. The comparison is based on both random formulae, and a set of formulae that are either in common use or found in the literature. In both cases, it is significantly more efficient in terms of number of states, transitions, acceptance conditions, and strength of the resulting automaton.

Etessami and Holzmann [8] have independently developed a similar approach. In their approach, like ours, rewriting is followed by translation and simulation-based optimization. Etessami and Holzmann perform rewriting using a set of rules that is incomparable to ours. They do not expand on the translation stage. In the minimization stage backward simulation is not employed. Their technique is geared towards non-generalized automata.

The flow of this paper is as follows. Section 2 presents the preliminaries. Section 3 covers rewriting of the LTL formulae. Section 4 describes how boolean optimization can be used to make the translation into automata more efficient. Then, Sections 5 and 6 describe the simplification of the automaton by deleting arcs and states, and pruning acceptance conditions. Finally, Section 7 presents the experimental results, and Section 8 concludes the paper.

2 Preliminaries

There are several variants of Büchi automata. We adopt automata with multiple acceptance conditions and with labels on the states (as opposed to labels on the arcs) as in [9].

Definition 1. *A* labeled, generalized Büchi automaton *is a six-tuple*

$$\mathcal{A} = \langle Q, Q_0, \delta, \mathcal{F}, D, \mathcal{L} \rangle,$$

where Q is the finite set of states, $Q_0 \subseteq Q$ is the set of initial states, $\delta : Q \to 2^Q$ is the transition relation, $\mathcal{F} \subseteq 2^Q$ is the set of acceptance conditions (or fair sets), D is a finite domain, and $\mathcal{L} : Q \to 2^D$ is the labeling function.

A run *of $\mathcal{A}$ is an infinite sequence $\rho = \rho_0, \rho_1, \ldots$ over Q, such that $\rho_0 \in Q_0$, and for all $i \geq 0$, $\rho_{i+1} \in \delta(\rho_i)$. A run ρ is* accepting *if for each $F_i \in \mathcal{F}$ there exists $q_j \in F_i$ that appears infinitely often in ρ.*

The automaton accepts an infinite word $\sigma = \sigma_0, \sigma_1, \ldots$ in D^ω if there exists an accepting run ρ such that, for all $i \geq 0$, $\sigma_i \in \mathcal{L}(\rho_i)$. The language of $\mathcal{A}$, denoted by $L(\mathcal{A})$, is the subset of D^ω accepted by $\mathcal{A}$.

We write $\mathcal{A}^q$ for the labeled, generalized Büchi automaton $\langle Q, \{q\}, \delta, \mathcal{F}, D, \mathcal{L}\rangle$.

Notice that with labels on the states we must allow multiple initial states. Multiple acceptance conditions, although not strictly necessary, simplify the translation. We will refer to a labeled generalized Büchi automaton simply as a Büchi automaton.

The temporal logic LTL is obtained from propositional logic by adding three temporal operators: U (until), R (releases, the dual of U), and X (next). The familiar G and F are defined as abbreviations, as are $\mathbf{T}$ (true) and $\mathbf{F}$ (false). Translation of an LTL formula φ into a Büchi automaton is accomplished by application of expansion rules also known as *tableau rules*:

$$\psi_1 \,\mathsf{U}\, \psi_2 = \psi_2 \vee (\psi_1 \wedge \mathsf{X}\,(\psi_1 \,\mathsf{U}\, \psi_2)) \ , \qquad \psi_1 \,\mathsf{R}\, \psi_2 = \psi_2 \wedge (\psi_1 \vee \mathsf{X}\,(\psi_1 \,\mathsf{R}\, \psi_2)) \ .$$

The rules are applied to φ until the resulting expression is a propositional formula in terms of *elementary subformulae* of φ. An elementary formula is a constant, an atomic proposition, or a formula starting with X. The expanded formula, put in disjunctive normal form (DNF), is an *elementary cover* of φ. Each term of the cover identifies a state of the automaton. The atomic propositions and their negations in the term define the *label* of the state, that is, the conditions that the input word must satisfy in that state. The remaining elementary subformulae of the term form the *next part* of the term; they are LTL formulae that identify the obligations that must be fulfilled to obtain an accepting run; they determine the transitions out of the state as well as the acceptance conditions.

The expansion process is applied to the next part of each state, creating new covers until no new obligations are produced. In this way, a *closed* set of elementary covers is obtained. The set is closed in the sense that there is an elementary cover in the set for the next part of each term of each cover in the set. The automaton is obtained by connecting each state to the states in the cover for its next part. The states in the elementary cover of φ are the initial states. Acceptance conditions are added to the automaton for each elementary subformula of the form $\mathsf{X}\,(\psi_1 \,\mathsf{U}\, \psi_2)$. The acceptance condition contains all the states s such that the label of s does not imply $\psi_1 \,\mathsf{U}\, \psi_2$ or the label of s implies ψ_2.

3 Rewriting the Formula

The first step towards an efficient translation is rewriting, a cheap, simple, and effective way to minimize result of the translation. Prior to generation of the Büchi automaton, a formula is put in positive normal form. Then we use the following identities and their duals to rewrite the formula, always replacing the left-hand side by the right-hand side.

$\varphi \leq \psi \Rightarrow (\varphi \wedge \psi) \equiv \varphi$
$\varphi \leq \neg\psi \Rightarrow (\varphi \wedge \psi) \equiv \mathbf{F}$
$(\mathsf{X}\,\varphi)\ \mathsf{U}\ (\mathsf{X}\,\psi) \equiv \mathsf{X}\,(\varphi\ \mathsf{U}\ \psi)$
$(\varphi\ \mathsf{R}\ \psi) \wedge (\varphi\ \mathsf{R}\ r) \equiv \varphi\ \mathsf{R}\ (\psi \wedge r)$
$(\varphi\ \mathsf{R}\ r) \vee (\psi\ \mathsf{R}\ r) \equiv (\varphi \vee \psi)\ \mathsf{R}\ r$
$(\mathsf{X}\,\varphi) \wedge (\mathsf{X}\,\psi) \equiv \mathsf{X}\,(\varphi \wedge \psi)$
$\mathsf{X}\,\mathbf{T} \equiv \mathbf{T}$
$\varphi\ \mathsf{U}\ \mathbf{F} \equiv \mathbf{F}$
$\varphi \leq \psi \Rightarrow (\varphi\ \mathsf{U}\ \psi) \equiv \psi$
$\neg\psi \leq \varphi \Rightarrow (\varphi\ \mathsf{U}\ \psi) \equiv (\mathbf{T}\ \mathsf{U}\ \psi)$

$\mathsf{G}\,\mathsf{F}\,\varphi \vee \mathsf{G}\,\mathsf{F}\,\psi \equiv \mathsf{G}\,\mathsf{F}\,(\varphi \vee \psi)$
$\mathsf{F}\,\mathsf{X}\,\varphi \equiv \mathsf{X}\,\mathsf{F}\,\varphi$
$\varphi \leq \psi \Rightarrow \varphi\ \mathsf{U}\ (\psi\ \mathsf{U}\ r) \equiv \psi\ \mathsf{U}\ r$
$\psi\ \mathsf{U}\ \mathsf{G}\,\mathsf{F}\,\varphi \equiv \mathsf{G}\,\mathsf{F}\,\varphi$
$\psi\ \mathsf{R}\ \mathsf{G}\,\mathsf{F}\,\varphi \equiv \mathsf{G}\,\mathsf{F}\,\varphi$
$\mathsf{X}\,\mathsf{G}\,\mathsf{F}\,\varphi \equiv \mathsf{G}\,\mathsf{F}\,\varphi$
$\mathsf{F}\,(\varphi \wedge \mathsf{G}\,\mathsf{F}\,\psi) \equiv (\mathsf{F}\,\varphi) \wedge (\mathsf{G}\,\mathsf{F}\,\psi)$
$\mathsf{G}\,(\varphi \vee \mathsf{G}\,\mathsf{F}\,\psi) \equiv (\mathsf{G}\,\varphi) \vee (\mathsf{G}\,\mathsf{F}\,\psi)$
$\mathsf{X}\,(\varphi \wedge \mathsf{G}\,\mathsf{F}\,\psi) \equiv (\mathsf{X}\,\varphi) \wedge (\mathsf{G}\,\mathsf{F}\,\psi)$
$\mathsf{X}\,(\varphi \vee \mathsf{G}\,\mathsf{F}\,\psi) \equiv (\mathsf{X}\,\varphi) \vee (\mathsf{G}\,\mathsf{F}\,\psi)$

The rewriting rules have been chosen to eliminate redundancies and to reduce the size of the resulting automaton. Checking for $\varphi \leq \psi$ is hard in general. Hence, we just look for simple cases that can be detected by purely syntactic means. We use the following set of rules and their duals.

$\varphi \leq \varphi$
$\varphi \leq \mathbf{T}$
$(\varphi \leq \psi) \wedge (\varphi \leq \chi) \Rightarrow \varphi \leq (\psi \wedge \chi)$
$(\varphi \leq \chi) \vee (\psi \leq \chi) \Rightarrow (\varphi \wedge \psi) \leq \chi.$

$\chi \leq \psi \Rightarrow \chi \leq (\varphi\ \mathsf{U}\ \psi)$
$(\varphi \leq \chi) \wedge (\psi \leq \chi) \Rightarrow (\varphi\ \mathsf{U}\ \psi) \leq \chi$
$(\varphi \leq \chi) \wedge (\psi \leq s) \Rightarrow (\varphi\ \mathsf{U}\ \psi) \leq (\chi\ \mathsf{U}\ s)$

The first two rules are the terminal cases of a recursive procedure in which one applies the remaining ones to decompose the problem.

Example 1. The rewriting rules transform the formula $(\mathsf{X}\,p\ \mathsf{U}\ \mathsf{X}\,q) \vee \neg\mathsf{X}\,(p\ \mathsf{U}\ q)$ into $\mathbf{T}$. The automaton produced by our algorithm when rewriting is disabled is shown in Figure 1. The optimal automaton obviously has only one state with a self-loop.

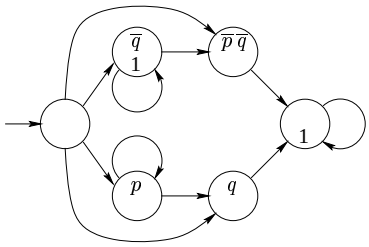

Fig. 1. Sub-optimal automaton for $(\mathsf{X}\,p\ \mathsf{U}\ \mathsf{X}\,q) \vee \neg\mathsf{X}\,(p\ \mathsf{U}\ q)$. Each node is annotated with its label, (first line), and the fair sets to which it belongs (second line). In the label, an overline indicates negation, and concatenation indicates conjunction.

4 Reducing the Number of States via Boolean Optimization

The LTL formula produced by rewriting is the input to the second phase of the procedure, which produces a closed set of elementary covers and constructs the Büchi

automaton from it. We refer to the covers in this set as the elementary covers of the automaton. An LTL formula has infinitely many elementary covers; which one is chosen affects the size of the resulting automaton by directly affecting what states are added to the automaton, and by determining what covers will belong to the closed set.

By regarding each elementary formula as a literal, one can apply boolean optimization techniques to minimize the cost of the elementary covers. (In this case, the implication tests required to compute the fair sets must also be carried out with boolean techniques.) However, the best result is not always obtained by computing a minimum cost cover for each formula. The following example illustrates this case.

Example 2. Consider translating the LTL formula $\varphi = \mathsf{F}\, p \wedge \mathsf{F}\, \neg p$ into a Büchi automaton. The algorithm of [5] produces the automaton on the left of Figure 2. Applying the

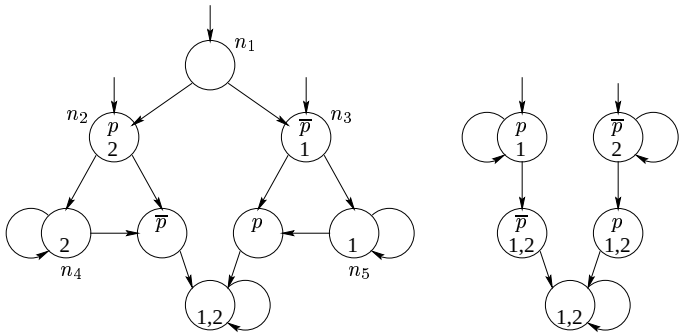

Fig. 2. Büchi automata for $\mathsf{F}\, p \wedge \mathsf{F}\, \neg p$: without (left) and with (right) boolean optimization.

tableau rules to φ yields $(p \wedge \mathsf{X}\,\mathsf{F}\, \neg p) \vee (\neg p \wedge \mathsf{X}\,\mathsf{F}\, p) \vee (\mathsf{X}\,\mathsf{F}\, p \wedge \mathsf{X}\,\mathsf{F}\, \neg p)$. The three terms of this cover correspond to the three initial states of the left automaton of Figure 2 (n_1, n_2, and n_3). The third term of the disjunctive normal form is the consensus of the first two; hence, it can be dropped. As a result, State n1 is removed. Applying the tableau rules to $\mathsf{F}\, p$ and $\mathsf{F}\, \neg p$ yields $\mathsf{F}\, p = p \vee \mathsf{X}\,\mathsf{F}\, p$ and $\mathsf{F}\, \neg p = \neg p \vee \mathsf{X}\,\mathsf{F}\, \neg p$, from which the remaining five states of the automaton on the left of Figure 2 are produced. Notice, however, that the alternative expansion $\mathsf{F}\, p = p \vee (\neg p \wedge \mathsf{X}\,\mathsf{F}\, p)$, $\mathsf{F}\, \neg p = \neg p \vee (p \wedge \mathsf{X}\,\mathsf{F}\, \neg p)$, though it requires more literals, prevents the creation of States n_4 and n_5, leading to the automaton on the right of Figure 2.

We can formulate the choice of the optimum covers as a 0-1 ILP as follows. We define one set of formulae $\Phi(\psi_0)$, and a set of terms $\Gamma(\psi_0)$ such that the automaton for ψ_0 will be obtained from elementary covers for a subset of the formulae in $\Phi(\psi_0)$, and will have states corresponding to a subset of $\Gamma(\psi_0)$. We then impose constraints that guarantee that the automaton will recognize exactly the models of the LTL formula.

Definition 2. *For an LTL formula φ, let $E(\varphi)$ be the expansion of φ in terms of elementary subformulae. Let $M(\varphi)$ be the set of minterms of $E(\varphi)$, and $P(\varphi)$ be the*

set of prime implicants of $E(\varphi)$. Finally, for γ a term of an elementary cover, let $N(\gamma) = \bigwedge\{\psi : \mathsf{X}\,\psi$ is a literal of $\gamma\}$ be its next part.

The sets $\Phi(\psi_0)$ and $\Gamma(\psi_0)$ are the smallest sets that satisfy the following constraints:

$$\psi_0 \in \Phi(\psi_0)\ , \qquad \Psi \subseteq \Phi(\psi_0) \Rightarrow P(\bigwedge \Psi) \subseteq \Gamma(\psi_0)\ ,$$
$$\gamma \in \Gamma(\psi_0) \Rightarrow N(\gamma) \in \Phi(\psi_0)\ .$$

We associate 0-1 variables to the elements of $\Phi(\psi_0)$ and $\Gamma(\psi_0)$ as follows: $y_i = 1$ if and only if an elementary cover of ψ_i is a cover of the automaton; $x_i = 1$ if and only if γ_i is a state of the automaton. We then search for $\min \sum x_i$, subject to:

$$y_0 \wedge \bigwedge_{\psi_i \in \Phi} (\neg y_i \vee \bigwedge_{\mu_j \in M(\psi_i)} \bigvee_{\gamma_k \in \Gamma} (x_k \wedge [\mu_j \leq \gamma_k \leq \psi_i])) \wedge \bigwedge_{\gamma_i \in \Gamma} (\neg x_i \vee y_{I(N(\gamma_i))})\ ,$$

where $I(\psi_j) = j$. The intuition behind this formulation is that constructing the automaton corresponds to finding DNF formulae for a set of boolean functions. (Each function is an elementary cover for some LTL formula.) It is well known that the solution consists of prime implicants of intersections of subsets of the functions in the set [15]. A function needs to be in the set if it is the expansion of the given LTL formula, or if it is the expansion of the next part of a term in the elementary cover chosen for a function that is in the set. This situation gives rise to *closure* constraints analogous to those found in the minimization of incompletely specified finite state machines [10]. Notice that we do not guarantee the optimum Büchi automaton for the given LTL formula (cf. Example 5 in Section 6); rather, we guarantee that no closed set of elementary covers can be found that has fewer terms, and such that each elementary cover can be generated from the formula it covers by exclusive application of the tableau rules and the laws of boolean algebra. Note that prior research on translation algorithms has focused on this class of algorithms.

Example 3. Continuing Example 2, let $\psi_0 = \mathsf{F}\,p \wedge \mathsf{F}\,\neg p$. We find:

$$\begin{array}{lllll} \psi_0 = \mathsf{F}\,p \wedge \mathsf{F}\,\neg p & \Rightarrow & \gamma_0 = p \wedge \mathsf{X}\,\mathsf{F}\,\neg p, & \gamma_1 = \neg p \wedge \mathsf{X}\,\mathsf{F}\,p, & \gamma_2 = \mathsf{X}\,\mathsf{F}\,p \wedge \mathsf{X}\,\mathsf{F}\,\neg p \\ \psi_1 = \mathsf{F}\,p & \Rightarrow & \gamma_3 = p, & \gamma_4 = \mathsf{X}\,\mathsf{F}\,p & \\ \psi_2 = \mathsf{F}\,\neg p & \Rightarrow & \gamma_5 = \neg p, & \gamma_6 = \mathsf{X}\,\mathsf{F}\,\neg p. & \end{array}$$

Since $\psi_0 = \psi_1 \wedge \psi_2$, no further formulae and terms are generated by considering subsets of $\Phi(\psi_0)$. The minterms of the formulae in $\Phi(\psi_0)$ are:

$$\begin{aligned} M(\psi_0) = \{&p \wedge \mathsf{X}\,\mathsf{F}\,p \wedge \mathsf{X}\,\mathsf{F}\,\neg p, p \wedge \neg\mathsf{X}\,\mathsf{F}\,p \wedge \mathsf{X}\,\mathsf{F}\,\neg p, \\ &\neg p \wedge \mathsf{X}\,\mathsf{F}\,p \wedge \mathsf{X}\,\mathsf{F}\,\neg p, \neg p \wedge \mathsf{X}\,\mathsf{F}\,p \wedge \neg\mathsf{X}\,\mathsf{F}\,\neg p\} \\ M(\psi_1) = \{&p \wedge \mathsf{X}\,\mathsf{F}\,p \wedge \mathsf{X}\,\mathsf{F}\,\neg p, p \wedge \mathsf{X}\,\mathsf{F}\,p \wedge \neg\mathsf{X}\,\mathsf{F}\,\neg p, p \wedge \neg\mathsf{X}\,\mathsf{F}\,p \wedge \mathsf{X}\,\mathsf{F}\,\neg p, \\ &p \wedge \neg\mathsf{X}\,\mathsf{F}\,p \wedge \neg\mathsf{X}\,\mathsf{F}\,\neg p, \neg p \wedge \mathsf{X}\,\mathsf{F}\,p \wedge \mathsf{X}\,\mathsf{F}\,\neg p, \neg p \wedge \mathsf{X}\,\mathsf{F}\,p \wedge \neg\mathsf{X}\,\mathsf{F}\,\neg p\} \\ M(\psi_2) = \{&\neg p \wedge \mathsf{X}\,\mathsf{F}\,p \wedge \mathsf{X}\,\mathsf{F}\,\neg p, \neg p \wedge \mathsf{X}\,\mathsf{F}\,p \wedge \neg\mathsf{X}\,\mathsf{F}\,\neg p, \neg p \wedge \neg\mathsf{X}\,\mathsf{F}\,p \wedge \mathsf{X}\,\mathsf{F}\,\neg p, \\ &\neg p \wedge \neg\mathsf{X}\,\mathsf{F}\,p \wedge \neg\mathsf{X}\,\mathsf{F}\,\neg p, p \wedge \mathsf{X}\,\mathsf{F}\,p \wedge \mathsf{X}\,\mathsf{F}\,\neg p, p \wedge \neg\mathsf{X}\,\mathsf{F}\,p \wedge \mathsf{X}\,\mathsf{F}\,\neg p\}\ . \end{aligned}$$

The constraint is:

$$
\begin{aligned}
& y_0 \wedge \\
& [\neg y_0 \vee (x_0 \vee x_2) \wedge x_0 \wedge (x_1 \vee x_2) \wedge x_1] \wedge \\
& [\neg y_1 \vee (x_0 \vee x_2 \vee x_3 \vee x_4) \wedge (x_3 \vee x_4) \wedge (x_0 \vee x_3) \wedge x_3 \wedge (x_1 \vee x_2 \vee x_4) \wedge (x_1 \vee x_4)] \wedge \\
& [\neg y_2 \vee (x_1 \vee x_2 \vee x_5 \vee x_6) \wedge (x_1 \vee x_5) \wedge (x_5 \vee x_6) \wedge x_5 \wedge (x_0 \vee x_2 \vee x_6) \wedge (x_0 \vee x_6)] \wedge \\
& (\neg x_0 \vee y_2) \wedge (\neg x_1 \vee y_1) \wedge (\neg x_2 \vee y_0) \wedge (\neg x_4 \vee y_1) \wedge (\neg x_6 \vee y_2) \ ,
\end{aligned}
$$

which simplifies to $y_0 \wedge y_1 \wedge y_2 \wedge x_0 \wedge x_1 \wedge x_3 \wedge x_5$. The feasible assignment that minimizes $\sum_{0 \leq i \leq 6} x_i$ is: $y_0 = y_1 = y_2 = x_0 = x_1 = x_3 = x_5 = 1$, $x_2 = x_4 = x_6 = 0$. This assignment corresponds to the solution found in Example 2. The fifth state of Figure 2 is required because $N(p) = N(\neg p) = \mathbf{T}$.

Solving the 0-1 ILP exactly is expensive, and, as we just observed, only guarantees optimality within a class of algorithms; therefore, we adopt a heuristic approach. In choosing the elementary cover for a formula, we first obtain a prime and irredundant cover from the one produced by the approach of [5]. Then, we look for existing states that imply the new terms. If we find a state that exactly matches a new term, we do not need to create a new state for the latter. If we find an existing state that implies the new term, we try to *reduce* [3] the new term to the existing one. That is, we check whether the replacement of the new term by the existing one changes the function represented by the cover. For instance, in Example 2, the initial cover for $\mathsf{F}\, p$, $p \vee \mathsf{X}\, \mathsf{F}\, p$, is reduced to $p \vee (\neg p \wedge \mathsf{X}\, \mathsf{F}\, p)$, because the second term already appears in the cover for φ.

If, on the other hand, we find a term that is implied by the new term, we check whether it can be reduced to the new term in all covers in which it already appears. We impose the constraint that the reduction only add atomic propositions or their negations. This constraint leaves the next part of the reduced term unchanged, and does not invalidate the covers obtained up to that point.

5 Simplifying Büchi Automata Using Simulations

The Büchi automata produced by our translation algorithm are not necessarily optimal. In this section we examine criteria that allow us to remove states from an automaton, while retaining its language. The techniques presented in this section can be used to simplify arbitrary Büchi automata. They do not always find the smallest possible Büchi automaton. Rather, they minimize the number of states and transitions heuristically. We deal with the simplification of the acceptance conditions in Section 6.

Our results derive from the notions of direct and reverse simulation. Direct simulation relations for Büchi automata have been studied in [6], and used in [1] for state-space minimization. Raimi [17] uses both direct and reverse simulations to minimize the state space, but does not take fairness into account.

In Subsection 5.3, we contrast simulation with language containment. Intuitively, simulation takes care of correspondence of acceptance conditions, which is one of the reasons why it is stronger than language containment. We show that this means that the conditions under which we can use simulation to reduce the number of states are more general than the conditions under which we can use language containment.

It is clear that we can remove all states that are not reachable from at least one initial state. Such states are not produced by the translation procedure, but they do occur as a result of other optimizations. It is equally obvious that we can remove any state q with $\mathcal{L}(q) = \emptyset$ or $\delta(q) = \emptyset$.

Definition 3. *A* direct simulation relation *over the states of a Büchi automaton $\mathcal{A}$ is any relation $\preceq \subseteq Q \times Q$ that satisfies the following property:*

$$p \preceq q \text{ implies } \begin{cases} \mathcal{L}(p) \subseteq \mathcal{L}(q) \ , \ \ F \in \mathcal{F} \Rightarrow [p \in F \Rightarrow q \in F] \\ s \in \delta(p) \Rightarrow \exists t \in \delta(q) : s \preceq t \ . \end{cases}$$

The largest direct simulation relation is denoted by $\preceq_D$. If both $p \preceq_D q$ and $q \preceq_D p$, then p and q are direct-simulation equivalent ($p \simeq_D q$).

A reverse simulation relation *over the states of $\mathcal{A}$ is any relation $\preceq \subseteq Q \times Q$ that satisfies the following property:*

$$p \preceq q \text{ implies } \begin{cases} \mathcal{L}(p) \subseteq \mathcal{L}(q) \ , \ \ p \in Q_0 \Rightarrow q \in Q_0 \ , \ \ F \in \mathcal{F} \Rightarrow [p \in F \Rightarrow q \in F] \\ s \in \delta^{-1}(p) \Rightarrow \exists t \in \delta^{-1}(q) : s \preceq t \ . \end{cases}$$

The largest reverse simulation relation is denoted by $\preceq_R$. If both $p \preceq_R q$ and $q \preceq_R p$, then p and q are reverse-simulation equivalent ($p \simeq_R q$).

The largest direct and reverse simulation relations can be found in polynomial time as the greatest fixpoints of the recursive definitions. Our definition of direct simulation corresponds to that of *BSR-aa* of [6]. If $p \preceq_D q$ in $\mathcal{A}$, then $L(\mathcal{A}^p) \subseteq L(\mathcal{A}^q)$ [16, 6].

5.1 Direct Simulation

A simulation relation between two states may allow us to remove a transition without disturbing the simulation relation.

Theorem 1. *Let $\mathcal{A}$ be a Büchi automaton. For $p, q \in Q$, $p \neq q$, assume that $p \preceq_D q$. Let $\mathcal{M} = \langle Q, Q_{0M}, \delta_M, \mathcal{F}, D, \mathcal{L} \rangle$, where $Q_{0M} = Q_0 \setminus \{p\}$ if $q \in Q_0$, and $Q_{0M} = Q_0$ otherwise, and δ_M is defined as follows:*

$$\delta_M(s) = \begin{cases} \delta(s) \setminus \{p\} & \text{if } q \in \delta(s), \\ \delta(s) & \text{otherwise.} \end{cases}$$

Then for all $s, t \in Q$, $s \preceq_D t$ in $\mathcal{A}$ if and only if $s \preceq_D t$ in $\mathcal{M}$. Also, any state in $\mathcal{A}$ is simulation-equivalent to the state of the same name in $\mathcal{M}$.

Proof. (Sketch.) For the first statement, one can prove that the largest simulation relation on $\mathcal{A}$ is a simulation relation on $\mathcal{M}$, using Definition 3 and transitivity of $\preceq_D$. In the same manner, one can prove that the largest simulation relation on $\mathcal{M}$ is a simulation relation on $\mathcal{A}$. This proves that $s \preceq_D t$ in $\mathcal{A}$ if and only if $s \preceq_D t$ in $\mathcal{M}$. The second statement can also be proved by direct application of Definition 3. □

The following corollary holds because simulation implies language equivalence.

Corollary 1. *Let $\mathcal{A}$ and $\mathcal{M}$ be as in Theorem 1. Then, $L(\mathcal{A}) = L(\mathcal{M})$.*

Theorem 1 obviously works in both directions. Hence, we can also add an arc from r to p if $q \in \delta(r)$ and $p \preceq_D q$. Repeated application of this transformation gives us the following corollary, which implies that we can remove one of any two simulation-equivalent states.

Corollary 2. *Let $\mathcal{A}$ be a Büchi automaton. Let $p, q \in Q$, $p \neq q$, and assume that $p \simeq_D q$. Let $\mathcal{M} = \langle Q, Q_{0M}, \delta_M, \mathcal{F}, D, \mathcal{L}\rangle$, where $Q_{0M} = (Q_0 \setminus \{p\}) \cup \{q\}$ if $p \in Q_0$, and $Q_{0M} = Q_0$ otherwise, and δ_M coincides with δ, except that, for all s, if $p \in \delta(s)$, $\delta_M(s) = (\delta(s) \setminus \{p\}) \cup \{q\}$. Then $L(\mathcal{A}) = L(\mathcal{M})$, and for all $s, t \in Q$, we have $s \preceq_D t$ in $\mathcal{A}$ if and only if $s \preceq_D t$ in $\mathcal{M}$.*

5.2 Reverse Simulation

In this subsection, we present techniques similar to the one in the last subsection, but pertaining to reverse similarity.

Theorem 2. *Let $\mathcal{A}$ be a Büchi automaton. For $p, q \in Q$, $p \neq q$, assume that $p \preceq_R q$. Let $\mathcal{M} = \langle Q, Q_0, \delta_M, \mathcal{F}, D, \mathcal{L}\rangle$, where δ_M coincides with δ, except that $\delta_M(p) = \delta(p) \setminus \delta(q)$. Then for all $s, t \in Q$, $s \preceq_R t$ in $\mathcal{A}$ if and only if $s \preceq_R t$ in $\mathcal{M}$, and any state in $\mathcal{A}$ is reverse-simulation equivalent to the state of the same name in $\mathcal{M}$.*

Corollary 3. *Let $\mathcal{A}$ and $\mathcal{M}$ be as in Theorem 2. Then, $L(\mathcal{A}) = L(\mathcal{M})$.*

Proof. (Sketch.) For any accepting run ρ of $\mathcal{A}$, we can construct an accepting run ρ' of $\mathcal{M}$. We can do this because for every i, we can choose a state ρ'_i in $\mathcal{M}$ such that $\rho_i \preceq_R \rho'_i$, $\rho'_i \in Q_0$ for $i = 0$, and $\rho'_i \in \delta(\rho'_{i-1})$ for $i > 0$. □

In analogy to direct simulation, we only need to retain one state in every reverse-similarity equivalence class.

Corollary 4. *Let $\mathcal{A}$ be a Büchi automaton. Let $p, q \in Q$, $p \neq q$, and assume that $p \simeq_R q$. Let $\mathcal{M} = \langle Q, Q_0, \delta_M, \mathcal{F}, D, \mathcal{L}\rangle$, where $\delta_M(p) = \emptyset$, $\delta_M(q) = \delta(p) \cup \delta(q)$, and $\delta_M(s) = \delta(s)$ for $s \notin \{p, q\}$. Then $L(\mathcal{A}) = L(\mathcal{M})$, and for all $s, t \in Q$ we have $s \preceq_R t$ in $\mathcal{A}$ if an only if $s \preceq_R t$ in $\mathcal{M}$.*

5.3 Language Containment

In this section we consider the more general case of language containment, and contrast it with simulation relations. The techniques in this section are not used in the algorithm, since language inclusion can not be checked easily, and because the minimization requires stricter conditions in the case of language containment. Indeed, we cannot simply substitute $L(\mathcal{A}^p) \subseteq L(\mathcal{A}^q)$ for $p \preceq_D q$ in Theorem 1 as evidenced by the following example.

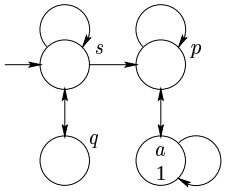

Fig. 3. Automaton showing that $L(\mathcal{A}^p) = L(\mathcal{A}^q)$ is not sufficient to allow simplification.

Example 4. Consider the automaton of Figure 3. The languages $L(\mathcal{A}^p)$ and $L(\mathcal{A}^q)$ are the same (they correspond to the formula $\mathsf{GF}\, a$). State s is a common predecessor of p and q. However, we cannot remove the arc from s to p without making the language of the automaton empty. The problem is that the accepting runs starting at q must use the arc that we want to remove in order to reach the accepting state. Notice that q does not direct simulate p.

The next theorem is analogous to Corollary 1 for direct simulation. It allows us to remove an arc from a state r to a successor p if r has another successor q with $L(\mathcal{A}^p) \subseteq L(\mathcal{A}^q)$. We impose a condition sufficient to guarantee that the $L(\mathcal{A}^q)$ is not changed by removal of the arc.

Theorem 3. *Let $\mathcal{A}$ be a Büchi automaton. For $p, q \in Q$, $p \neq q$, assume that $L(\mathcal{A}^p) \subseteq L(\mathcal{A}^q)$. Let $\mathcal{M} = \langle Q, Q_{0M}, \delta_M, \mathcal{F}, D, \mathcal{L}\rangle$, where $Q_{0M} = Q_0 \setminus \{p\}$ if $q \in Q_0$, and $Q_{0M} = Q_0$ otherwise, and δ_M coincides with δ, except that for all $r \in Q$ such that $q \in \delta(r)$ and r is not reachable from q, $\delta_M(r) = \delta(r) \setminus \{p\}$. Then $L(\mathcal{A}) = L(\mathcal{M})$.*

By repeated application of Theorem 3, we can also prove that we can merge two language-equivalent states p and q, as long as q cannot reach any predecessors of p, in analogy to Corollary 2.

6 Pruning the Fair Sets

The automata produced by the algorithm of Section 4 have as many accepting conditions as there are *until* subformulae in the LTL formula. In this section we show how to shrink some fair sets or drop them altogether. Simplifying the fair sets has several benefits. First of all, it may lead to a reduction in strength of the resulting automaton. (For instance, if the automaton is reduced to *terminal*, model checking requires a simple reachability analysis.) Even if the strength of the automaton is not reduced, fewer, smaller fair sets usually lead to faster convergence of the language emptiness check. Simplifying the acceptance conditions may also enable further reductions in the number of states or transitions. Finally, the resulting automaton is often easier to understand.

The pruning of the fair sets is based on the analysis of the strongly connected components (SCCs) of the state graph.

Definition 4. *An SCC of a Büchi automaton $\mathcal{A}$ is* fair *if it is non-trivial and it intersects all fair sets. Let Θ_A be the union of all fair SCCs of $\mathcal{A}$; Θ_A is called the* final *set of $\mathcal{A}$.*

Every accepting run of an automaton is eventually contained in a fair SCC. Hence, states not in Θ_A can be removed from the accepting sets. Furthermore, states with no paths to Θ_A can be removed altogether: their language is empty. Fair sets that contain Θ_A, or that include another fair set, can be dropped without changing the language accepted by the automaton. It is not uncommon for one fair sets to become included in another once a few states are dropped from it, as a consequence, for instance, of the application of the following results.

When one fair set is contained in another we can remove states from the latter if they do not appear in the former. We can extend this result by focusing on a single SCC.

Theorem 4. *Let $\mathcal{A} = \langle Q, Q_0, \delta, \mathcal{F}, D, \mathcal{L}\rangle$ be a Büchi automaton and let γ be an SCC of $\mathcal{A}$. Suppose that there exist $F, F' \in \mathcal{F}$ such that $F \cap \gamma \subseteq F' \cap \gamma$. Let $\mathcal{M} = \langle Q, Q_0, \delta, \mathcal{F}_M, D, \mathcal{L}\rangle$, where*

$$\mathcal{F}_M = (\mathcal{F} \setminus \{F'\}) \cup \{F'_M\}, \text{ with}$$
$$F'_M = F' \setminus (\gamma \setminus F).$$

Then $L(\mathcal{A}) = L(\mathcal{M})$.

The next theorem, which shows that a state can be removed from a fair set if there is a suitable 'detour', will be followed by an example.

Theorem 5. *Let $\mathcal{A} = \langle Q, Q_0, \delta, \mathcal{F}, D, \mathcal{L}\rangle$ be a Büchi automaton. Let F be a fair set in $\mathcal{F}$, and p, q distinct states in F such that $\mathcal{L}(p) \subseteq \mathcal{L}(q)$, $\delta(p) \subseteq \delta(q)$, and $\delta^{-1}(p) \subseteq \delta^{-1}(q)$. Let $\mathcal{M} = \langle Q, Q_0, \delta, \mathcal{F}_M, D, \mathcal{L}\rangle$, where $\mathcal{F}_M = (\mathcal{F} \setminus \{F\}) \cup \{F \setminus \{p\}\}$. Then $L(\mathcal{A}) = L(\mathcal{M})$.*

Example 5. Two automata for $\mathsf{G}\,(\mathsf{F}\,p \wedge \mathsf{F}\,q)$ are shown in Figure 4. Theorem 5 applies to

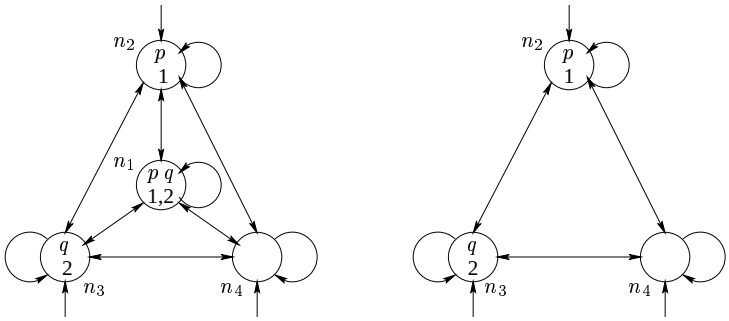

Fig. 4. Büchi automata for $\mathsf{G}\,(\mathsf{F}\,p \wedge \mathsf{F}\,q)$ illustrating the application of Theorem 5.

State n_1 of the automaton on the left. After its removal from the two fair sets, State n_1 can be merged with State n_4. The resulting automaton is shown on the right of Figure 4. It is worth pointing out that the automaton on the left is constructed from a minimum

cost elementary cover of $\mathsf{G}(\mathsf{F}\,p \wedge \mathsf{F}\,q)$. Since there is no elementary cover with fewer than four terms, there is no way to generate the three-state solution by direct application of the translation algorithm of Section 4.

Definition 5. *Let $p \preceq_r q$ be like $p \preceq_R q$, but without the condition on the fair sets.*

Theorem 6. *Let $\mathcal{A} = \langle Q, Q_0, \delta, \mathcal{F}, D, \mathcal{L}\rangle$ be a Büchi automaton. Let $\gamma \subseteq Q$ be an SCC, and $q \in \gamma$ a state. Suppose that for all $p \in \gamma$ we have $p \preceq_r q$, $\delta(p) \subseteq \delta(q)$, and $q \in \delta(p)$. Let $\mathcal{M} = \langle Q, Q_0, \delta_M, \mathcal{F}, D, \mathcal{L}\rangle$ with $\delta_M(s) = \delta(s) \setminus (\gamma \setminus \{q\})$ for $s \in \gamma$, $s \neq q$, and $\delta_M(s) = \delta(s)$ otherwise. Then, $L(\mathcal{A}) = L(\mathcal{M})$.*

Proof. Clearly, $L(\mathcal{A}) \supseteq L(\mathcal{M})$. Let ρ be an accepting run for $\sigma \in D^\omega$ in $\mathcal{A}$. If ρ does not go through γ, it is also an accepting run for σ in $\mathcal{M}$. If ρ enters and exits γ, let (s, p) be the last arc of ρ before it leaves γ. If $s \notin \gamma$, ρ does not use any arc internal to γ. Hence, it is also an accepting run for σ in $\mathcal{M}$. If $s \in \gamma$, $s \preceq_r q$. (This is not changed by the removal of arcs internal to γ.) Hence, there is a run ρ_M in $\mathcal{M}$ that reaches q when ρ reaches s, goes to p (also a successor of q) next, and then continues identical to ρ.

If ρ dwells in γ forever ($\inf(\rho) \subseteq \gamma$), γ is a fair SCC (it intersects all fair sets). For each $F_i \in \mathcal{F}$, choose $s_i \in (F_i \cap \inf(\rho))$. We can build a run ρ_M such that every occurrence of s_i is followed by an occurrence of $s_{(i+1) \bmod |\mathcal{F}|}$. This can be done by going from s_i to q, waiting in q until ρ goes to $s_{(i+1) \bmod |\mathcal{F}|}$, and then going to that state. We may "skip a beat" if $s_{(i+1) \bmod |\mathcal{F}|}$ follows immediately s_i in ρ, but we shall just take the next occurrence of $s_{(i+1) \bmod |\mathcal{F}|}$. Since $p \preceq_r q$ for all $p \in \gamma$, $\mathcal{L}(p) \subseteq \mathcal{L}(q)$; hence, ρ_M accepts σ. □

Thanks to Theorem 6 we can remove the arcs out of States n_2 and n_3 in the right automaton of Figure 4, except those going to State n_4. It is also possible to remove n_2 and n_3 from the set of initial states, though this is not covered by the theorem.

The next result is based on the observation that, in determining what states of an SCC should belong to a fair set, we can ignore the arcs out of the SCC.

Theorem 7. *Let $\mathcal{A} = \langle Q, Q_0, \delta, \mathcal{F}, D, \mathcal{L}\rangle$ be a Büchi automaton. Let γ be an SCC of $\mathcal{A}$, and p a state in γ such that, when the arcs out of γ are removed, $q \in \gamma$ implies $q \preceq_D p$ and $q \preceq_r p$. Let $\mathcal{M} = \langle Q, Q_0, \delta, \mathcal{F}_M, D, \mathcal{L}\rangle$, where $\mathcal{F}_M = \{F \setminus (\gamma \setminus \{p\}) | F \in \mathcal{F}\}$. Then $L(\mathcal{A}) = L(\mathcal{M})$.*

If every cycle through a state p visits a fair set in some other state, we do not need to include p in that fair set.

Theorem 8. *Let $\mathcal{A} = \langle Q, Q_0, \delta, \mathcal{F}, D, \mathcal{L}\rangle$ be a Büchi automaton. Let $F \in \mathcal{F}$, and p a state in F such that every cycle through p intersects F in a state different from p. Let $\mathcal{M} = \langle Q, Q_0, \delta, \mathcal{F}_M, D, \mathcal{L}\rangle$, where $\mathcal{F}_M = (\mathcal{F} \setminus F) \cup \{F \setminus \{p\}\}$. Then $L(\mathcal{A}) = L(\mathcal{M})$.*

After application of the results presented in this section, a Büchi automaton may still have multiple fair sets. It is well-known that we can always reduce the number of acceptance conditions to one by introducing a counter. This is not usually done because the algorithm of Emerson and Lei [7] deals with multiple acceptance conditions. However, for weak and terminal automata, we do not need to resort to the counter to reduce the number of fair sets to one.

Definition 6. *A Büchi automaton is* weak *if and only if for each SCC γ, either for each fair set F, $\gamma \subseteq F$, or there exists a fair set F such that $\gamma \cap F = \emptyset$. A Büchi automaton is* terminal *if and only if it is weak, there is no arc from a fair SCC to a non-fair SCC, and for every state s in a fair SCC, $\bigcup\{\mathcal{L}(t) : t \in \delta(s)\} = D$.*

Theorem 9. *Let $\mathcal{A} = \langle Q, Q_0, \delta, \mathcal{F}, D, \mathcal{L}\rangle$ be a weak (generalized) Büchi automaton. Let $\mathcal{M} = \langle Q, Q_0, \delta, \{\Theta_A\}, D, \mathcal{L}\rangle$. Then $L(\mathcal{A}) = L(\mathcal{M})$.*

States that are not in Θ_A can be added to fair sets as long as Θ_A is not changed by the addition. In particular, states in trivial SCCs are "don't care" states when it comes to checking the conditions on $\mathcal{F}$ in Definition 3.

The automaton simplification procedure prunes the fair sets a first time before applying simulation-based simplification, because smaller fair sets tend to produce larger simulation relations. The simulation-based techniques may break up SCCs because they remove arcs. Hence, pruning is performed a second time after they are applied. This can be iterated until a fixpoint is reached, but the benefits are minor and the extra CPU time comparatively large.

7 Experimental Results

In this section we report the results obtained with a translator from LTL to Büchi automata named *Wring* and based on the results discussed thus far. *Wring* is written in Perl. Based on our experience, we estimate that speed could be increased by at least an order of magnitude by coding the algorithms in C. However, CPU times are modest, and even a relatively slow implementation is more than adequate.

Table 1 shows a comparison to the algorithms analyzed in [5] on common formulae and formulae found in the literature. It should be noted that the outcomes of the translation algorithms depend on the order in which sub-formulae are examined. Hence, different implementations may produce different results. Rewriting of the formula (see Section 3) is most effective in detecting tautologies and contradictions. The transformations discussed in Section 6 are quite effective in reducing the number of fair sets.

Table 2 shows results for randomly generated formulae [5], showing the effects of each of the three major extensions that we propose. It should be noted that working on simpler formulae and smaller automata offsets most of the additional cost of minimization. One can see from Table 2 that the reduction in transitions and fair sets corresponds to an increase in the number of terminal automata. Though we do not present a detailed analysis of the dependence of the results on the statistics of the formulae (number of nodes, number of atomic propositions, and percentage of temporal operators), we have observed the same trends reported in [5].

8 Conclusions and Future Work

We have presented a heuristic algorithm for the generation of small Büchi automata from LTL formulae. It works in three stages: rewriting of the formula, boolean optimization to reduce the number of states in the translation procedure, and simplification

Table 1. Comparison to our implementation of the algorithms analyzed in [5]. For each formula and each method, the numbers of states, transitions, and accepting conditions are given. The letters in the accepting sets columns indicate whether the automata are strong (s), weak (w), or terminal (t).

	GPVW			GPVW+			LTL2AUT			Wring		
Formula	st.	tran.	acc.	st.	tran.	acc.	st.	tran.	acc.	st.	tran.	acc.
$p \mathsf{U} q$	3	4	1t	3	4	1t	3	4	1t	3	4	1t
$p \mathsf{U} (q \mathsf{U} r)$	6	10	2t	6	10	2t	4	7	2t	4	7	1t
$\neg(p \mathsf{U} (q \mathsf{U} r))$	7	15	0w	7	15	0w	7	15	0w	6	10	0w
$\mathsf{G}\mathsf{F} p \rightarrow \mathsf{G}\mathsf{F} q$	9	15	2s	5	8	2s	5	11	2s	4	7	1s
$\mathsf{F} p \mathsf{U} \mathsf{G} q$	8	15	2w	8	15	2w	6	17	2w	3	4	1w
$\mathsf{G} p \mathsf{U} q$	5	6	1w	5	6	1w	5	6	1w	5	6	1w
$\neg(\mathsf{F}\mathsf{F} p \leftrightarrow \mathsf{F} p)$	12	16	2t	1	1	2t	1	1	2t	0	0	0t
$\neg(\mathsf{G}\mathsf{F} p \rightarrow \mathsf{G}\mathsf{F} q)$	10	28	2s	6	16	2s	6	24	2s	3	6	1s
$\neg(\mathsf{G}\mathsf{F} p \leftrightarrow \mathsf{G}\mathsf{F} q)$	20	56	4s	12	32	4s	12	48	4s	5	11	1s
$p \mathsf{R} (p \vee q)$	5	11	0w	5	11	0w	4	8	0w	3	4	0w
$(\mathsf{X}(p) \mathsf{U} \mathsf{X}(q)) \vee \neg\mathsf{X}(p \mathsf{U} q)$	9	13	1w	9	13	1w	9	13	1w	1	1	0t
$(\mathsf{X}(p) \mathsf{U} q) \vee \neg\mathsf{X}(p \mathsf{U} (p \wedge q))$	10	21	1w	10	21	1w	9	17	1w	7	12	1w
$\mathsf{G}(p \rightarrow \mathsf{F} q) \wedge$ $((\mathsf{X}(p) \mathsf{U} q) \vee \neg\mathsf{X}(p \mathsf{U} (p \wedge q)))$	29	149	2s	23	118	2s	17	56	2s	10	23	1s
$\mathsf{G}(p \rightarrow \mathsf{F} q) \wedge$ $((\mathsf{X}(p) \mathsf{U} \mathsf{X}(q)) \vee \neg\mathsf{X}(p \mathsf{U} q))$	26	93	2s	24	78	2s	20	51	2s	3	8	1s
$\mathsf{G}(p \rightarrow \mathsf{F} q)$	5	17	1s	5	17	1s	3	8	1s	3	8	1s
$\neg\mathsf{G}(p \rightarrow \mathsf{X}(q \mathsf{R} r))$	5	8	2w	5	8	2w	5	8	2w	5	8	1t
$\neg(\mathsf{G}\mathsf{F} p \vee \mathsf{F}\mathsf{G} q)$	10	28	2s	6	16	2s	6	24	2s	3	6	1s
$\mathsf{G}(\mathsf{F} p \wedge \mathsf{F} q)$	4	16	2s	4	16	2s	8	40	2s	3	9	2s
$\mathsf{F} p \wedge \mathsf{F} \neg p$	11	21	2t	8	14	2t	8	14	2t	5	7	1t
$\mathsf{X}((((s \mathsf{U} p) \mathsf{R} r) \mathsf{U} (s \mathsf{R} r))$ $(\mathsf{G}(q \vee \mathsf{G}\mathsf{F} p) \wedge \mathsf{G}(r \vee \mathsf{G}\mathsf{F} \neg p))$ $\vee \mathsf{G} q \vee \mathsf{G} r$	130	1013	2s	78	483	2s	64	378	2s	8	13	1w
$(\mathsf{G}(q \vee \mathsf{F}\mathsf{G} p) \wedge \mathsf{G}(r \vee \mathsf{F}\mathsf{G} \neg p))$ $\vee \mathsf{G} q \vee \mathsf{G} r$	33	320	2s	23	234	2s	14	71	2s	4	6	2s
$(\mathsf{G}(q \vee \mathsf{F}\mathsf{G} p) \wedge \mathsf{G}(r \vee \mathsf{F}\mathsf{G} \neg p))$ $\vee \mathsf{G} q \vee \mathsf{G} r$	51	368	2w	29	170	2w	10	29	2w	2	2	0w
$\neg((\mathsf{G}(q \vee \mathsf{G}\mathsf{F} p) \wedge \mathsf{G}(r \vee \mathsf{G}\mathsf{F} \neg p))$ $\vee \mathsf{G} q \vee \mathsf{G} r)$	132	428	6w	106	310	6w	64	230	6w	11	26	1w
$\neg((\mathsf{G}(q \vee \mathsf{F}\mathsf{G} p) \wedge \mathsf{G}(r \vee \mathsf{F}\mathsf{G} \neg p))$ $\vee \mathsf{G} q \vee \mathsf{G} r)$	114	478	6s	78	288	6s	66	289	6s	21	32	1s
$\mathsf{G}(q \vee \mathsf{X}\mathsf{G} p) \wedge \mathsf{G}(r \vee \mathsf{X}\mathsf{G} \neg p)$	16	40	0w	12	28	0w	10	20	0w	5	7	0w
$\mathsf{G}(q \vee (\mathsf{X} p \wedge \mathsf{X} \neg p))$	2	2	0w	1	1	0w	1	1	0w	1	1	0w
$(p \mathsf{U} p) \vee (q \mathsf{U} p)$	9	13	2t	5	7	2t	5	7	2t	3	3	1t
Total	681	3204	51	484	1940	51	372	1397	51	131	231	22

Table 2. Results for 1000 random formulae with parse graphs of 15 nodes, using 3 atomic propositions, and uniform distribution of the operators ($\vee$, $\wedge$, X, U, R) for the internal nodes. In the designation of the method, 'r' stands for rewriting, 'b' stands for boolean optimization, and 's' stands for automaton simplification. Method Wring is equivalent to LTL2AUT+rbs. The experiments were run on an IBM Intellistation with a 400 MHz Pentium II CPU.

Method	states	transitions	fair sets	initial states	weak	terminal	CPU time (s)
GPVW	34216	137565	1555	4388	743	87	1195.0
GPVW+	29231	114146	1555	4246	745	95	1016.1
LTL2AUT	19424	62370	1555	3803	718	136	640.5
LTL2AUT+r	9430	25722	747	2117	511	374	209.7
LTL2AUT+b	13213	29573	1555	3161	738	162	1047.4
LTL2AUT+s	6063	11300	637	1628	362	552	1229.8
LTL2AUT+rb	7632	16124	747	1998	532	385	439.9
LTL2AUT+rs	5800	10624	491	1569	382	531	527.6
LTL2AUT+bs	5834	10525	620	1553	354	568	1292.2
Wring	5648	10053	492	1535	390	537	631.4

of the automaton. Rewriting is a cheap, simple, and effective technique to shrink the formula and reduce redundancy. The boolean optimization technique that we have presented yields the smallest translation of any procedure in its class. We do not know of any translation procedures that do not fall within this class, its most important characteristic being the use of the given expansion functions. Since the optimal algorithm is expensive, we have implemented a heuristic approximation. Finally, the simplification step uses direct and reverse simulation to minimize the number of states and transitions in the formula, and it prunes the acceptance conditions to make the emptiness check easier.

We have shown that our algorithm outperforms previously published algorithms on both random formulae, and formulae in common use and from the literature.

We are investigating weaker relations that allow for state minimization, and can still be computed efficiently. In particular, we are working on combinations of direct and reverse simulation, and simulation with less strict conditions on the acceptance conditions such as *BSR-lc* of [6]. Another area of investigation is the use of semantic information about the literals in the simplification of elementary covers. For symbolic model checking, the automata must be given binary encodings. A careful choice of the state encodings and the use of don't care conditions derived from simulation relations and from the analysis of the SCCs of the graph should help reduce the cost of the language emptiness check.

Acknowledgment

The authors thank Kavita Ravi for stimulating discussions on language containment and for her many comments on various drafts of this manuscript.

References

[1] A. Aziz, V. Singhal, G. M. Swamy, and R. K. Brayton. Minimizing interacting finite state machines. In *Proceedings of the International Conference on Computer Design*, pages 255–261, Cambridge, MA, October 1994.

[2] R. Bloem, K. Ravi, and F. Somenzi. Efficient decision procedures for model checking of linear time logic properties. In N. Halbwachs and D. Peled, editors, *Eleventh Conference on Computer Aided Verification (CAV'99)*, pages 222–235. Springer-Verlag, Berlin, 1999. LNCS 1633.

[3] R. K. Brayton, G. D. Hachtel, C. T. McMullen, and A. Sangiovanni-Vincentelli. *Logic Minimization Algorithms for VLSI Synthesis*. Kluwer Academic Publishers, Boston, Massachusetts, 1984.

[4] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98:142–170, 1992.

[5] M. Daniele, F. Giunchiglia, and M. Y. Vardi. Improved automata generation for linear time temporal logic. In N. Halbwachs and D. Peled, editors, *Eleventh Conference on Computer Aided Verification (CAV'99)*, pages 249–260. Springer-Verlag, Berlin, 1999. LNCS 1633.

[6] D. L. Dill, A. J. Hu, and H. Wong-Toi. Checking for language inclusion using simulation relations. In K. G. Larsen and A. Skou, editors, *Third Workshop on Computer Aided Verification (CAV'91)*, pages 255–265. Springer, Berlin, July 1991. LNCS 575.

[7] E. A. Emerson and C.-L. Lei. Efficient model checking in fragments of the propositional mu-calculus. In *Proceedings of the First Annual Symposium of Logic in Computer Science*, pages 267–278, June 1986.

[8] K. Etessami and G. J. Holzmann. Optimizing Büchi automata. In *Proc. 11th International Conference on Concurrency Theory (CONCUR2000)*, pages 153–167. Springer, 2000. LNCS 1877.

[9] 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. Chapman & Hall, 1995.

[10] A. Grasselli and F. Luccio. A method for minimizing the number of internal states in incompletely specified sequential networks. *IEEE Transactions on Electronic Computers*, EC-14(3):350–359, June 1965.

[11] Y. Kesten, Z. Manna, H. McGuire, and A. Pnueli. A decision algorithm for full propositional temporal logic. In C. Courcoubetis, editor, *Fifth Conference on Computer Aided Verification (CAV '93)*, pages 97–109, Berlin, 1993. Springer-Verlag. LNCS 697.

[12] Y. Kesten, A. Pnueli, and L.-o. Raviv. Algorithmic verification of linear temporal logic specifications. In *International Colloquium on Automata, Languages, and Programming (ICALP-98)*, pages 1–16, Berlin, 1998. Springer. LNCS 1443.

[13] O. Kupferman and M. Y. Vardi. Freedom, weakness, and determinism: From linear-time to branching-time. In *Proc. 13th IEEE Symposium on Logic in Computer Science*, June 1998.

[14] O. Lichtenstein and A. Pnueli. Checking that finite state concurrent programs satisfy their linear specification. In *Proceedings of the Twelfth Annual ACM Symposium on Principles of Programming Languages*, pages 97–107, New Orleans, January 1985.

[15] E. J. McCluskey, Jr. Minimization of Boolean functions. *Bell Syst. Technical Journal*, 35:1417–1444, November 1956.

[16] R. Milner. An algebraic definition of simulation between programs. *Proc. 2nd Int. Joint Conf. on Artificial Intelligence*, pages 481–489, 1971.

[17] R. S. Raimi. *Environment Modeling and Efficient State Reachability Checking*. PhD thesis, University of Texas at Austin, 1999.

[18] P. Wolper, M. Y. Vardi, and A. P. Sistla. Reasoning about infinite computation paths. In *Proceedings of the 24th IEEE Symposyium on Foundations of Computer Science*, pages 185–194, 1983.