Electronic Notes in Theoretical Computer Science 66 No. 2 (2002)
URL: http://www.elsevier.nl/locate/entcs/volume66.html 15 pages

Simple and Efficient Translation from LTL Formulas to Büchi Automata

Xavier Thirioux [1]

IRIT - LIMA, 2 rue Camichel, 31071 Toulouse, France,
Xavier.Thirioux@enseeiht.fr

Abstract

We present a collection of simple on-the-fly techniques to generate small Büchi automata from Linear Time Logic formulas. These techniques mainly involve syntactic characterizations of formulas, and yet allow efficient computations. Thus heavily relying on such proof-theoretic issues, we can omit the classical formula pre-simplification step, and also simulation-based post-simplification steps (aka model-theoretic issues).

Although closely related to other similar recent works in the same topic, our ideas have led to an implementation that performs significantly better than some of the best available tools, such as Wring or LTL2BA. We compare our tool BAOM ("Büchi Automata Once More") with others, on formulas commonly found in the literature, and on randomly generated testbenchs.

Key words: Linear Time Logic. Büchi automata. Tableaux-based method. Syntactic characterizations of formulas.

Introduction

This paper describes several new techniques to implement an efficient translation from Linear Time Logic (LTL) specifications to Büchi automata. Our prime motivation was to implement a new symbolic BDD-based [2] model-checker [BCM+92] based upon Linear Time Logic specifications for synchronous programs, especially those written in Esterel. This work extends previous works on the Xeve model-checker [Bou97].

In our symbolic framework, a well-known solution to achieve this goal is first to translate (the negation of) any given LTL formula into an observer (a finite state machine) that is plugged into the original design we want to check. Then, by symbolic forward image computations of the product system, find

[1] This work was partially supported by the SYNTEL RNRT project, France.
[2] BDD stands for Binary Decision Diagram.

an execution that meets the fairness side conditions imposed by the formula, the so-called Büchi conditions. Though the principle is rather simple, an efficient implementation that avoids exponential blow-ups during translation of a formula into an observer is difficult to achieve. Recently, some promising new algorithms have been found to lower the size of automata. The resulting tools, namely LTL2BA, Wring and EqLTL (see respectively [GO01,SB00,EH00]) seemingly outperform SPIN [Hol97].

Our main contribution is to provide a new implementation (called BAOM) that behaves linearly (in space and time) for many more formulas (especially formulas containing fairness constraints), where previous algorithms exhibit exponential blow-ups. Moreover, we obtain automata that are in the average smaller than with any other method. Our prototype directly builds a Generalized Büchi Automaton from a Linear Temporal Logic formula, without the need for intermediate data-structures. For matters of efficiency, our transitions are labelled with BDDs rather than with single atomic propositions. Also, our automata are generalized in a mixed sense, i.e. we have fairness conditions on states as well as on transitions, depending upon the shape of the formula. Notice that our ideas are primarily concerned with on-the-fly optimizations based upon syntactic relations between formulas. We also introduce a notion of "merged" states in our automata, in order to factorize sub-tableaux when possible and reduce further the number of states.

Nevertheless, our algorithm widely borrows ideas from recent tableaux methods, from recent techniques around alternating automata, and also from syntactic relations between formulas. Actually, we include the syntactic simplification rules of [SB00] on-the-fly during the tableaux generation, and generalize in this way similar rules of [DGV99] as well as [GO01]. Some of these rules can also be seen as an cheap alternative to the "boolean optimization" paradigm of [SB00], which is a general solution to remove redundant and complementary sub-formulas occurring in tableaux. In our case, this simplification may introduce new fairness constraints on transitions, as in [GO01].

Related works come in many flavours, but are principaly concerned with improvements of the tableau method described in [VW94] and [GPVW95]. In [EH00] the authors present an algorithm in three steps : first a rewriting step, followed by a standard translation and finally a simulation-based optimization. In [SB00], the same kind of techniques are applied, yet with totally different rewriting rules and with simulation relations that can be computed more efficiently. In [GO01], the authors also reuse the same set of rewriting rules as in [SB00], and consider very simple on-the-fly simplification rules that avoid fixpoint computations necessary in simulation-based methods. The simplification process, though simple, is still efficient due to a specific translation based upon alternating automata where fairness constraints exclusively concern transitions.

Finally, all these works point out the ability to simplify in some cases the

fairness constraints of the SCCs [3] of the generated automata, for instance by the recursive removing of the unfair terminal SCCs.

The roadmap : section 1 introduces preliminary notions and reminds the original tableaux method. Section 2 presents our new revisited tableaux algorithm that builds a first version of a Büchi automaton from a LTL formula. This algorithm splits in four parts :

(i) Use modified tableaux rules to generate a basic automaton (section 2.1);

(ii) Normalize and simplify transitions (section 2.2);

(iii) Detect unfair SCCs and simplify the automaton (section 2.3);

(iv) Merge transition-equivalent states (section 2.4).

Thereafter, section 3 presents a classical post-simplification phase to reduce the number of states. Finally, we show in section 4 some promising results of our prototype and compare them with other similar tools, and then we conclude in section 5.

1 Preliminaries

We define here a variant of Büchi automata, also called Generalized Büchi Automata, with multiple acceptance conditions on the states as well as on the transition edges. Labels are located on the transition edges, and are boolean formulas (denoted below as $\mathcal{B}(AP)$) built from a set of atomic propositions AP([4]). We make use of a data structure to represent the edges (thus replacing the traditional δ function) because we actually need to distinguish between different transition edges with different fairness constraints and compatible labels. Notice that we treat uniformly state and transition fairness.

Definition 1.1 *A GBA is a five-tuple :*

$$\mathcal{A} = \langle AP, Q, Q_0, E, \mathcal{F} \rangle$$

where AP is the set of atomic propositions, Q is the finite set of states, $Q_0 \subseteq Q$ is the set of initial states, $E \subseteq Q \times \mathcal{B}(AP) \times Q$ is the set of edges, and $\mathcal{F} \subseteq \mathcal{B}(Q \cup E)$ is the set of acceptance conditions, expressed as logical constraints. A (generalized) transition function $\delta \in \mathcal{B}(AP) \rightarrow 2^Q \rightarrow 2^Q$ can be recovered from E as :

$$\delta(p, qs) = \{q' \mid \exists q, l, q'.q \in qs \wedge \langle q, l, q' \rangle \in E \wedge \models p \rightarrow l\}$$

As p and l are encoded as BDD, we can easily decide whether $\models p \rightarrow l$ holds or not.

A run of $\mathcal{A}$ is an infinite sequence $\sigma = \langle q_0, i_0, t_0 \rangle; \langle q_1, i_1, t_1 \rangle; \ldots$ where $q_k \in Q$, $t_k \in E$ and $i_k \subseteq AP$, such that for all $k \geq 0$:

$$t_k = \langle q_k, l_k, q_{k+1} \rangle \wedge i_k \models l_k$$

[3] SCC stands for Strongly Connected Component.

[4] For X a set of ground terms, $\mathcal{B}(X)$ denotes its boolean closure.

A run σ is accepting if for each $F \in \mathcal{F}$, we have infinitely many k's such that :

$$t_{\mathrm{k}}, q_{\mathrm{k}+1} \models F$$

Finally, an automaton $\mathcal{A}$ accepts an infinite word of input events $i = i_0, i_1, \ldots$ over $(2^{AP})^{\omega}$, whenever there exists an accepting run of $\mathcal{A}$:

$$\sigma = \langle q_0, i_0, t_0 \rangle; \langle q_1, i_1, t_1 \rangle; \ldots$$

Its language $\mathcal{L}(\mathcal{A})$ is the set of infinite words it accepts.

Multiple initial states and multiple acceptance conditions are not mandatory, but are considered here only for convenience with respect to the overall model-checking process in which the translation step occurs.

Definition 1.2 *The linear time logic (LTL) is built from propositional logic by adding temporal operators, yielding the following syntax :*

$$\begin{aligned} LTL ::= \; & AP \\ | \; & \mathcal{B}(LTL) \\ | \; & LTL \; \mathsf{U} \; LTL \\ | \; & LTL \; \mathsf{R} \; LTL \\ | \; & \bigcirc \; LTL \end{aligned}$$

$\bigcirc$ is the "next-time" operator, U is the (strong) "until" operator and R is the "release" operator. R and U are dual of each other. Usual $\Box$ ("always") and $\Diamond$ ("eventually") operators are defined as $\Box\phi = False \; \mathsf{R} \; \phi$ and $\Diamond\phi = True \; \mathsf{U} \; \phi$.

We briefly recall here the standard tableaux method [VW94,GPVW95], as we use it as a basis for our own extension. Each state of the automaton denotes and identifies a LTL formula in negative normal form, i.e. where negation has been pushed down the parse tree of the formula. Then, from a given state, the transition function is computed by means of semantic expansion rules. These rules consist in applying from left to right the following equalities to the state-formula, through the expansion function Exp :

$$\begin{aligned} Exp(prop) &= prop \\ Exp(\bigcirc\phi) &= \bigcirc\phi \\ Exp(\phi \vee \psi) &= Exp(\phi) \vee Exp(\psi) \\ Exp(\phi \wedge \psi) &= Exp(\phi) \wedge Exp(\psi) \\ Exp(\phi \; \mathsf{U} \; \psi) &= Exp(\psi \vee (\phi \wedge \bigcirc(\phi \; \mathsf{U} \; \psi))) \\ Exp(\phi \; \mathsf{R} \; \psi) &= Exp(\psi \wedge (\phi \vee \bigcirc(\phi \; \mathsf{R} \; \psi))) \end{aligned}$$

Thus, starting with a formula ϕ, we first expand it and then put it into disjunctive normal form. Each conjunctive term $\psi = \psi_1 \wedge \psi_2 \ldots$ will constitute a next state. According to the above rules, each ψ_{i} is either an atomic proposi-

tion ap_{i} or a next-time formula $\bigcirc\theta_{\mathrm{i}}$. Hence, the ψ-state will be labelled by a formula $\theta_1 \wedge \theta_2 \ldots$ whereas the transition edge from the ϕ-state to the ψ-state will be labelled by a propositional formula $ap_1 \wedge ap_2 \ldots$.

As usual, transitions labelled with unsatisfiable propositions are removed, thus removing unreachable states as well.

The initial states are built from expansion of the root formula. Multiple initial states can be avoided if the initial formula is not expanded. This may increase or decrease the number of states, depending upon the formula (see theorem 2.13).

Finally, as for the Büchi acceptance conditions, for each $\phi \mathsf{U} \psi$ occurring in state-formulas, there exists an acceptance formula $Fair_{\phi \mathsf{U} \psi}$ on states :

$$Fair_{\phi \mathsf{U} \psi} \ = \ \bigvee \{q \in Q \mid \phi \mathsf{U} \psi \notin q \ \vee \ \psi \in q\}$$

Then, for this particular kind of formulas on states, the condition for a run to be accepted boils down to the following statement : for each set $Fair_{\phi \mathsf{U} \psi}$, we have infinitely many q_k's such that $q_k \models Fair_{\phi \mathsf{U} \psi}$.

2 Tableaux method revisited

In the remainder, we propose different steps aiming at reducing the size of automata. All these improvements are relative to a pervasive automaton $\mathcal{M} = \langle AP, Q, Q_0, E, \mathcal{F} \rangle$ assumed at each step to be the result of previous transformations. To ease the description of our method, we define a notion of substitution on sets as :

$$S[e' \mid e] = \begin{cases} (S \setminus \{e\}) \cup \{e'\} \text{ if } e \in S \\ S \text{ else} \end{cases}$$

2.1 Expanding tableaux rules

When designing this new algorithm, our main goal was to obtain small and deterministic automata from a standard tableaux-based method.

We now define a temporal approximation of a LTL formula, driven by a positive integer, denoted as $\lceil \phi \rceil^d$. This approximation is a formula representing exactly the finite d-prefixes of infinite words identified by ϕ.

Definition 2.1 *For any $\phi \in$LTL under negative normal form and $d \geq 0$, we*

define the function $\lceil\phi\rceil^d$ *as below :*

$$\begin{aligned}
\lceil ap\rceil^d &= ap\\
\lceil \phi \vee \psi\rceil^d &= \lceil\phi\rceil^d \vee \lceil\psi\rceil^d\\
\lceil \phi \wedge \psi\rceil^d &= \lceil\phi\rceil^d \wedge \lceil\psi\rceil^d\\
\lceil \phi \ \mathsf{U}\ \psi\rceil^d &= \lceil\psi \vee (\phi \wedge \bigcirc(\phi \ \mathsf{U}\ \psi))\rceil^d\\
\lceil \phi \ \mathsf{R}\ \psi\rceil^d &= \lceil\psi \wedge (\phi \vee \bigcirc(\phi \ \mathsf{R}\ \psi))\rceil^d\\
\lceil \bigcirc\phi\rceil^0 &= True\\
\lceil \bigcirc\phi\rceil^{d+1} &= \bigcirc\lceil\phi\rceil^d
\end{aligned}$$

Lemma 2.2 *For any* $\phi \in LTL$ *and any* $d \geq 0$*, we have :* $\phi \Rightarrow \lceil\phi\rceil^d$ *and also* $\phi \vee (\lceil\neg\phi\rceil^d \wedge \psi) \Leftrightarrow \phi \vee \psi$*.*

Proof (sketch) The implication is proved by structural induction on ϕ. The equivalence then follows.

We can now modify the expansion rules taking into account this finite approximation. Indeed, given any integer d, we can use the following new rules for U and R, where $\neg\psi$ and $\neg\phi$ are put in negative normal form :

$$Exp(\phi \ \mathsf{U}\ \psi) = Exp(\psi \vee (\phi \wedge (\lceil\neg\psi\rceil^d \wedge \bigcirc(\phi \ \mathsf{U}\ \psi))))$$

$$Exp(\phi \ \mathsf{R}\ \psi) = Exp(\psi \wedge (\phi \vee (\lceil\neg\phi\rceil^d \wedge \bigcirc(\phi \ \mathsf{R}\ \psi))))$$

Theorem 2.3 *For any* $\phi \in LTL$ *and any* $d \geq 0$*, let* $\mathcal{M}_\mathrm{d}$ *be the automaton produced using revised expansion rules, then* $\mathcal{L}(\mathcal{M}) = \mathcal{L}(\mathcal{M}_\mathrm{d})$*.*

Proof (sketch) The automata are produced according to semantic expansion rules so that they exactly accept words denoted by state-formulas. Then lemma 2.2 is used to convert revised formulas to standard ones, proving that $\mathcal{M}$ and $\mathcal{M}_\mathrm{d}$ accept the same language.

For our specific usage, using a great value of d may increase the number of different states, as the $\lceil.\rceil^d$ operator generates new next-time sub-formulas. Though, due to the extra constraints upon transitions introduced by the prefix formulas, this may also increase the deterministic flavour of the resulting automaton, that is we obtain at an early stage many more incompatible transitions, which later on we won't have to compare (see theorem 2.9).

After some conclusive experiments showing that the average number of states tend to increase as d does, we decided for the time being to restrain our choice to $d = 0$, leading to a good balance between a small overhead and a better overall performance. For instance, with formulas under the form $\bigwedge_{\mathrm{i}=1\ldots\mathrm{N}} \Box\Diamond p_\mathrm{i}$, regarding N as a parameter, our method can save upto an exponential number of states with respect to the Wring tool, or proceed exponentially faster than the LTL2BA tool (we obtain the same automaton in

this case [5]).

We now assume that each state denotes a conjunctively interpreted set of formulas, instead of a single conjunctive formula.

Let $\phi \leq \psi$ be a relation of syntactic implication between two formulas, similar to the ones presented in [SB00] and [DGV99]. This relation will greatly help us in reducing the size of automata. Notice that this relation doesn't expand temporal operators, so that its computational cost is moderate.

Definition 2.4 *For any $\phi, \psi \in LTL$, we define the relation $\phi \leq \psi$ as the smallest fixpoint of the following rules :*

$False \leq \psi$	$\phi \leq True$
$\phi_1 \vee \phi_2 \leq \psi \Leftarrow \phi_1 \leq \psi \wedge \phi_2 \leq \psi$	$\phi \leq \psi_1 \wedge \psi_2 \Leftarrow \phi \leq \psi_1 \wedge \phi \leq \psi_2$
$\phi \leq \psi_1 \vee \psi_2 \Leftarrow \phi \leq \psi_1 \vee \phi \leq \psi_2$	$\phi_1 \wedge \phi_2 \leq \psi \Leftarrow \phi_1 \leq \psi \vee \phi_2 \leq \psi$
$\phi_1 \mathsf{R} \phi_2 \leq \psi \Leftarrow \phi_2 \leq \psi$	$\phi \leq \psi_1 \mathsf{U} \psi_2 \Leftarrow \phi \leq \psi_2$
$\phi_1 \mathsf{R} \phi_2 \leq \psi_1 \mathsf{R} \psi_2 \Leftarrow \phi_1 \leq \psi_1 \wedge \phi_2 \leq \psi_2$	$\phi_1 \mathsf{U} \phi_2 \leq \psi_1 \mathsf{U} \psi_2 \Leftarrow \phi_1 \leq \psi_1 \wedge \phi_2 \leq \psi_2$

This definition allows us to remove weak formulas from states, and therefore to reduce in many cases the number of different states, by the mean of the following theorem.

Theorem 2.5 *Let $q = \{\phi_1, \phi_2, \ldots, \phi_n\} \in Q$ be a state such that $\phi_1 \leq \phi_2$. Let q' denote the set $\{\phi_2, \ldots, \phi_n\}$. Then $\mathcal{L}(\mathcal{M}) = \mathcal{L}(\mathcal{M}')$ with the automaton $\mathcal{M}' = \langle AP, Q', Q'_0, E', \mathcal{F}' \rangle$ defined below :*

- $Q' = Q[q' \mid q]$
- $Q'_0 = Q_0[q' \mid q]$
- $E' = \{\langle q_{\text{src}}, l, q_{\text{dst}}[q' \mid q] \rangle \mid \langle q_{\text{src}}, l, q_{\text{dst}} \rangle \in E\}$
- $\mathcal{F}' = \left\{ \begin{array}{l} \{\mathcal{F}air_\phi[q' \mid q] \mid \mathcal{F}air_\phi \in \mathcal{F}\} \quad \textit{if } \phi_1 \neq \llcorner \mathsf{U} \lrcorner \\ \mathcal{F}[\mathcal{F}air'_{\phi_1} \mid \mathcal{F}air_{\phi_1}] \quad \textit{else, with} \\ \quad \mathcal{F}air'_{\phi_1} = \mathcal{F}air_{\phi_1}[q' \mid q] \wedge (\neg q' \bigvee \{\langle q_{\text{src}}, l, q_{\text{dst}} \rangle \in E \mid q_{\text{dst}} \neq q\}) \end{array} \right\}$

Proof (sketch) As our implication relation is easily proved to be sound, the two automata accept the same language, disregarding fairness constraints. In the case the removed formula ϕ_1 is an until formula and thus involves a change in acceptance conditions, we report the fairness of ϕ_1 onto all the incoming transition edges of state q'. Hence we mimic the standard situation where fairness is on state q.

[5] As shown in test cases presented in section 4.

2.2 Normalizing transitions

Once the outgoing transition edges from a given state are built, we proceed with a normalization step in which we factor transitions. This factorization is possible (and simple) because we use BDDs to represent transition labels.

Definition 2.6 *Assuming that $\phi \in LTL$ is such that $\mathcal{F}air_\phi \in \mathcal{F}$ and $q, q' \in Q$, we define the following global fair (and unfair) labeling functions between two states :*

$$l_\phi(q,q') = \bigvee\{l \mid t = \langle q, l, q'\rangle \in E \wedge t, q' \models \mathcal{F}air_\phi\}$$
$$l_{\text{unfair}}(q,q') = \bigvee\{l \mid t = \langle q, l, q'\rangle \in E \wedge \forall \mathcal{F}air_\phi \in \mathcal{F}.t, q' \not\models \mathcal{F}air_\phi\}$$

With these functions, we can define our normalization step.

Theorem 2.7 *Let us define the automaton $\mathcal{M}' = \langle AP, Q, Q_0, E', \mathcal{F}'\rangle$:*

- $E' = \{\langle q, l, q'\rangle \mid l = l_\phi(q,q') \vee l = l_{\text{unfair}}(q,q')\}$
- $\mathcal{F}' = \{\mathcal{F}air'_\phi \mid \mathcal{F}air_\phi \in \mathcal{F}\}$ *with*
  $\mathcal{F}air'_\phi = \bigvee\{t \wedge q' \mid t = \langle q, l, q'\rangle \in E' \wedge l = l_\phi(q,q')\}$

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

Proof (sketch) This normalization may reduce the number of transition edges between two states[6] but has no effect on the language of $\mathcal{M}$ since we factorize edges with respect to fairness constraints. The only case to which we must pay attention is when a given transition edge is fair regarding at least two different constraints. Then we must split it into (at least) two different edges with the same label, each satisfying only one fair constraint. But for any accepted word, if an original edge of $\mathcal{M}$ would be triggered infinitely often, the resulting edges of $\mathcal{M}'$ could also be triggered infinitely often, each in turn. The converse also holds. So finally $\mathcal{L}(\mathcal{M}) = \mathcal{L}(\mathcal{M}')$.

Notice that in the above theorem, we can indeed easily simplify the fairness constraints, in order to keep only transition fairness. Actually, defining :

$$\mathcal{F}air'_\phi = \bigvee\{t \mid t = \langle q, l, q'\rangle \in E' \wedge l = l_\phi(q,q')\}$$

would also yield the same result. But we decided to keep both kinds of fairness information because this allows to implement simpler algorithms in our prototype.

The next step consists in trying to determinize transitions from any given state, i.e. to modify their labels so that their pairwise intersection becomes empty. This usually leads to a lesser number of (smaller) transition edges, and allow in practice further simplifications (see theorem 2.13). Besides, the structure of the automaton is then more easily amenable to efficient model-checking algorithms. Actually, the deterministic flavour of an automaton is a

[6] After this operation, there always exist less than $|\mathcal{F}|+1$ different transition edges between any two states.

salient feature in symbolic model-checking, because it appears to have a great influence on efficiency of partitioned states space exploration algorithms for instance.

Definition 2.8 *We classically extend the notion of implication between formulas to an implication between states. For any $q, q' \in Q$, we define :*

$$q \leq q' \ = \ \forall \phi' \in q'. \exists \phi \in q. \phi \leq \phi'$$

Theorem 2.9 *Let us consider $t_1 = \langle q, l_1, q_1 \rangle \in E$ and also $t_2 = \langle q, l_2, q_2 \rangle \in E$. Now assume $q_1 \leq q_2$, $l_1 \wedge l_2 \neq False$ and :*

$$\forall \mathcal{F}air_\phi \in \mathcal{F}. t_1, q_1 \models \mathcal{F}air_\phi \ \Rightarrow \ t_2, q_2 \models \mathcal{F}air_\phi$$

Then the automaton $\mathcal{M}' = \langle AP, Q, Q_0, E', \mathcal{F}' \rangle$ defined below :

- $E' = E[t'_1 \mid t_1]$ *with* $t'_1 = \langle q, l_1 \wedge \neg l_2, q_1 \rangle$
- $\mathcal{F}' = \mathcal{F}$

is such that $\mathcal{L}(\mathcal{M}) = \mathcal{L}(\mathcal{M}')$. We remove the new transition t'_1 from E' and set $\mathcal{F}' = \{\mathcal{F}air_\phi[False \mid t'_1] \mid \mathcal{F}air_\phi \in \mathcal{F}\}$ if $l_1 \wedge \neg l_2 = False$ holds.

Proof (sketch) Following definitions 2.4 and 2.8, $q_1 \leq q_2$ implies $\mathcal{L}(q_1) \subseteq \mathcal{L}(q_2)$. So, we can safely remove from t_1 the input events that are common with t_2. The fairness constraints don't need to be changed (but in case of mere removal) since it is easier to reach q_2 through t_2 than to reach q_1 through t_1 w.r.t. fairness constraints, and $l_1 \vee l_2 = (l_1 \wedge \neg l_2) \vee l_2$. Hence, if we had an accepted word passing from q to q_1 through $l_1 \wedge l_2$, we know that it would also be accepted via q_2.

2.3 Detecting unfair SCCs

Our last but one on-the-fly step can simplify the fairness constraints on states, by early detecting of certain unfair SCCs, i.e. SCCs where at least one fairness constraint is never satisfied for any of its states, or transient SCCs, i.e. SCCs with only one state and no self loop. Besides, we define a syntactic under-approximation of unfair and transient SCCs.

Definition 2.10 *For $q \in Q$, we define the following* FairLoop *and* Unstable *predicates*[7] *:*

$$\begin{aligned} \text{Unstable}(q) \ &= \ \exists \phi \in q. \phi \neq True \wedge \neg \text{FairLoop}(\phi, q) \\ \text{FairLoop}(\phi, q) \ &= \ \exists \psi = \psi_1 \mathsf{R} \psi_2 \in q. \phi = \psi \vee \phi \prec \psi_2 \end{aligned}$$

Lemma 2.11 *For any $q \in Q$, such that* Unstable(q)*, then the SCC of q is either transient or unfair.*

[7] For any formulas ϕ and ψ, $\phi \prec \psi$ denotes the subterm relation, but for negated atoms. That is, for p an atomic proposition, we have $p \not\prec \neg p$.

Proof (sketch) By contradiction. Assume the SCC of q is fair. Then, following the expansion rules, each non-R formula ϕ must be (transitively) generated by the right-hand side ψ of a R formula, which are the only fair looping operators in LTL. Hence, because we don't change the shape of formulas when expanding them, it is necessary to check $\phi \prec \psi$. Therefore, Unstable(q) does not hold. As a conclusion, an accepted run cannot remain stuck in the SCC of an unstable state.

Theorem 2.12 *Let q be an unstable state, we change the fairness constraints by defining the automaton $\mathcal{M}' = \langle AP, Q, Q_0, E, \mathcal{F}' \rangle$ as :*

- $\mathcal{F}' = \{\mathcal{F}air_\phi[False \mid q] \mid \mathcal{F}air_\phi \in \mathcal{F}\}$

Then we have $\mathcal{L}(\mathcal{M}) = \mathcal{L}(\mathcal{M}')$.

Proof (sketch) We can safely remove fairness information relative to an unstable state and its incoming transition edges, since an accepted word cannot visit it infinitely often, as proved by lemma 2.11. So, $\mathcal{L}(\mathcal{M}) = \mathcal{L}(\mathcal{M}')$.

2.4 Merging states

Last, but not least, we define the notion of merged states. It consists in merging some target states of some transitions with the same labels and fairness constraints, making a compound state. This also applies to the initial states that can be merged as one single state.

Theorem 2.13 *Let us consider $t_1 = \langle q, l_1, q_1 \rangle \in E$ and also $t_2 = \langle q, l_2, q_2 \rangle \in E$. Now assume we have $l_1 = l_2$([8]) and :*

$$\forall \mathcal{F}air_\phi \in \mathcal{F}. t_1, q_1 \models \mathcal{F}air_\phi \;\Leftrightarrow\; t_2, q_2 \models \mathcal{F}air_\phi$$

Then the automaton $\mathcal{M}' = \langle AP, Q', Q_0, E', \mathcal{F}' \rangle$ defined below :

- $Q' = Q \cup \{q_{12}\}$
- $E' = (E \setminus \{t_1, t_2\}) \cup \{t_{12}\}$ *with* $t_{12} = \langle q, l_1, q_{12} \rangle$
- $\mathcal{F}' = \{\mathcal{F}air_\phi[False \mid \{t_1, t_2\}] \vee \mathcal{F}air_\phi[t_{12} \wedge q_{12} \mid t_1 \wedge q_1] \mid \mathcal{F}air_\phi \in \mathcal{F}\}$

is such that $\mathcal{L}(\mathcal{M}) = \mathcal{L}(\mathcal{M}')$.

Proof (sketch) As we ensure that transitions as well as target states have the same impact on fairness, we can merge them without modifying the set of accepted words. Notice that the original target states are not removed, but if they become unreachable. Then, $\mathcal{L}(\mathcal{M}) = \mathcal{L}(\mathcal{M}')$.

A merged state is defined as the set of its components. In theorem 2.13, we have thus $q_{12} = \{q_1, q_2\}$. Ordinary states can also be defined as singleton sets. So from now on we move up a level and assert that a state indeed represents a set of sets of formulas.

[8] These labels are identical up to BDD normalization.

It seems likely that states accessible through the same label have something in common, and that it may be worth trying to identify them. We only consider identical labels, as if we would consider a more relaxed constraint (for instance labels with a non empty intersection), this could create exponentially more new states. In practice, it seems that most of the time interesting compound states are created, and not too many of them.

Nevertheless, it may happen that some merged state in the automaton is subsumed by its components, existing as states on their own. Then the merged state is there superfluous and can be safely removed.

The initial state, put in DNF, can also play the role of a merged state. By the following theorem, it can be split as any other real merged state in order to reduce the overall number of states.

For merged states to be split back, we have to define the notion of subsumption.

Definition 2.14 *For any set of sets of formulas (not necessarily a actual state) S, we define what it means to be subsumed by states of $\mathcal{M}$, with the following predicate :*

$$\begin{aligned} &\text{Subsumed}(S) &&\Leftarrow \exists q \in Q.S = q \\ &\text{Subsumed}(S_1 \cup S_2) &&\Leftarrow \text{Subsumed}(S_1) \wedge \text{Subsumed}(S_2) \end{aligned}$$

Theorem 2.15 *Let us consider $q = q_1 \cup \ldots \cup q_n \in Q$. Assume* Subsumed$(q)$ *holds. Then the automaton $\mathcal{M}' = \langle AP, Q', Q'_0, E', \mathcal{F}' \rangle$ defined below :*

- $Q' = Q \setminus \{q\}$
- $Q'_0 = \begin{cases} (Q_0 \setminus \{q\}) \cup \{q_1\} \cup \ldots \cup \{q_n\} & \text{if } q \in Q_0 \\ Q_0 & \text{else} \end{cases}$
- $E' = E \setminus \{\langle q_{\text{src}}, l, q \rangle \in E\}$
- $\mathcal{F}' = \{\mathcal{F}air_\phi[False \mid q] \mid \mathcal{F}air_\phi \in \mathcal{F}\}$

is such that $\mathcal{L}(\mathcal{M}) = \mathcal{L}(\mathcal{M}')$.

Proof (sketch) We remove a state q that is exactly subsumed by others as stated in definition 2.14, and redirect its incoming edges towards its components, which together recognize the same language as q. Henceforth $\mathcal{L}(\mathcal{M}) = \mathcal{L}(\mathcal{M}')$.

This theorem can be applied on-the-fly or later when the automaton is totally built. We chose to use it as soon as possible, in order to reduce the complexity of the post-simplification phase, but it may be worth postponing its use so as to process all merged states once and for all. Intermediate choices are currently being experimented too.

3 Post-simplification

As for automata post-simplification phase, we remove (all but one of) identical states, which is a pertaining step in all related works. Indeed, we haven't explored more general simulation relations, because they sometimes tend not to lend themselves to efficient computations.

Theorem 3.1 *Let us consider two states $q_1, q_2 \in Q$, with the same outgoing transition edges, i.e. such that for any $n = 1, 2$ and $\overline{n} = 3 - n$:*

$$\forall t_{\mathrm{n}} = \langle q_{\mathrm{n}}, l_{\mathrm{n}}, q' \rangle \in E. \exists t_{\overline{\mathrm{n}}} = \langle q_{\overline{\mathrm{n}}}, l_{\overline{\mathrm{n}}}, q' \rangle \in E.$$
$$l_{\mathrm{n}} = l_{\overline{\mathrm{n}}} \wedge \forall \mathcal{F}air_\phi \in \mathcal{F}. t_{\mathrm{n}}, q' \models \mathcal{F}air_\phi \;\Leftrightarrow\; t_{\overline{\mathrm{n}}}, q' \models \mathcal{F}air_\phi$$

Without loss of generality, we assume that whenever at least one of q_1 or q_2 is initial, then it is q_1. Then the automaton $\mathcal{M}' = \langle AP, Q', Q'_0, E', \mathcal{F}' \rangle$ defined below :

- $Q' = Q \setminus \{q_2\}$
- $Q'_0 = Q_0 \setminus \{q_2\}$
- $E' = \{\langle q_{\mathrm{src}}, l, q'_{\mathrm{dst}} \rangle \mid \langle q_{\mathrm{src}}, l, q_{\mathrm{dst}} \rangle \in E \wedge q'_{\mathrm{dst}} = q_{\mathrm{dst}}[q_1 \mid q_2]\}$
- $\mathcal{F}' = \{\mathcal{F}air_\phi[\langle q_1, l, q_{\mathrm{dst}} \rangle \mid \langle q_2, l, q_{\mathrm{dst}} \rangle \in E][q_1 \mid q_2] \mid \mathcal{F}air_\phi \in \mathcal{F}\}$ [9]

is such that $\mathcal{L}(\mathcal{M}) = \mathcal{L}(\mathcal{M}')$.

Proof (sketch) This is a classical operation in automata theory.

4 Experiments

The following examples have all been tested on a 400MHz bi-pentiumII PC, with 256 Mo.

We include first some tests taken from [EH00,GO01] (see figure 1), showing the relative performances of Wring, LTL2BA and BAOM. For the sake of simplicity, we decided to present only results of the best available tools, because they constantly outperform other tools like SPIN for instance. Likewise, elapsed time is most of the time not shown, because all these tools usually achieve the translation within 5 seconds, which is largely acceptable. Yet, there exist pathological formulas such that time consumption is then exponential. In this case, we precisely show elapsed time or indicate that a crash had occurred or timeout (10 hours) had expired (†).

Then, we show some results for 3 collections of 1000 randomly generated formulas, processed with LTL2AUT, Wring and BAOM. Temporal and boolean operators are drawn with the same probability (see figure 2).

Finally, in order to (loosely) compare BAOM to LTL2BA, though we had only a restricted access to the LTL2BA tool via a web page, we use a testbench

[9] In the above theorem, the term $\mathcal{F}air_\phi[\langle q_1, l, q_{\mathrm{dst}} \rangle \mid \langle q_2, l, q_{\mathrm{dst}} \rangle \in E]$ means that the substitution is performed for all $\langle q_2, l, q_{\mathrm{dst}} \rangle \in E$.

for our tool generated with the same hypothesis as a similar testbench for LTL2BA presented in [GO01] (see figure 3). We have randomly drawn 1000 generated formulas with 10 nodes and 3 atoms.

Our prototype is entirely written in OCaML [DU], and thus time comparisons with other tools issued from similar works is hardly relevant due to the extreme diversity of implementation languages (and computers). For instance, the SPIN tool [Hol97] as well as the LTL2BA tool [GO01] and the LTL2AUT tool [DGV99] are written in C, whereas the EqLTL tool [EH00] is written in ML, and the Wring tool [SB00] in Perl.

Notice that transient memory requirements are not mentionned here due to impracticability of measures. Besides, because our transformations are almost all applied on-the-fly, memory consumption in our case is linearly bound to the size of the resulting automaton.

LTL formulas	states(time in seconds)		
	Wring	LTL2BA	BAOM
examples from [EH00]			
$p\mathsf{U}(q \wedge \Box r)$	3	2	2
$p\mathsf{U}(q \wedge \bigcirc(r\mathsf{U}s))$	5	3	3
$p\mathsf{U}(q \wedge \bigcirc(r \wedge (\Diamond(s \wedge \bigcirc\Diamond(t \wedge \bigcirc\Diamond(u \wedge \bigcirc\Diamond v))))))$	13	7	7
$\Diamond(p \wedge \bigcirc\Box q)$	3	2	2
$\Diamond(p \wedge \bigcirc(q \wedge \bigcirc\Diamond r))$	6	4	4
$\Diamond(q \wedge \bigcirc(p\mathsf{U}r))$	5	3	3
$\Diamond\Box p \vee \Diamond\Box q$	4	3	3
$\Box(p \rightarrow q\mathsf{U}r)$	3	2	2
$\Diamond(p \wedge \bigcirc\Diamond(q \wedge \bigcirc\Diamond(r \wedge \bigcirc\Diamond s)))$	9	5	5
$\bigwedge_{i=1\ldots5} \Box\Diamond p_i$	31(195)	1	1
$(p\mathsf{U}(q\mathsf{U}r)) \vee (q\mathsf{U}(r\mathsf{U}p))$	4	5	2
$(p\mathsf{U}(q\mathsf{U}r)) \vee (q\mathsf{U}(r\mathsf{U}p)) \vee (r\mathsf{U}(p\mathsf{U}q))$	4	7	2
$\Box(p \rightarrow q\mathsf{U}(\Box r \vee \Box s))$	4	4	4
examples from [GO01]			
$\neg((\bigwedge_{i=1\ldots10} \Box\Diamond p_i) \rightarrow \Box(q \rightarrow \Diamond r))$	†	2(36000)	2(44)
$\neg(p_1\mathsf{U}(p_2\mathsf{U}(\ldots\mathsf{U}p_8)\ldots)$	†	8(1200)	8

Fig. 1. Examples excerpt from [EH00,GO01].

	10 nodes 3 atoms		15 nodes 3 atoms		20 nodes 5 atoms	
method	states	time	states	time	states	time
LTL2AUT	6698	127s	11086	453s	25528	2740s
Wring	4043	203s	4830	534s	7748	1973s
BAOM	3026	3.45s	3318	6.5s	4723	40s

Fig. 2. Comparison between LTL2AUT, Wring and BAOM.

method	formulas	avg. time	max. time	avg. states	max. states
LTL2BA	200	0.01	0.04	4.51	39
BAOM	1000	0.003	0.09	3.06	16

Fig. 3. Loose comparison between LTL2BA and BAOM.

5 Conclusion

We have succeeded in devising an efficient algorithm, based upon syntactic considerations, with techniques designed to be used on-the-fly. This shows that a careful examination of parse trees of formulas can lead to similar or better results than *a posteriori* simulation-based methods. In real-life applications, efficiency is also due to the heavy use of BDDs in our data-structures, but this advantage doesn't really show up in our test cases due to the small number of atomic propositions. The main original factors of improvement over other similar tools are the introduction of finite d-prefixes in our revised expansion rules and also the fairness paradigm we have developed in conjunction with the syntactic implication between formulas. Yet, a more careful study of the relationship between values of d, shape of formulas and size of resulting automatas should obviously be carried out. As for the merging states techniques, it appears to have an impact in case of quasi-redundant or incompatible sub-formulas (this is often the case for randomly generated formulas), but don't usually come into play for short hand-written specifications.

Notice that the current implementation of our tool hasn't been specifically geared towards efficiency since it was developped in a purely functional setting (except for the BDD package). More efficient techniques such as hash-caching (as used in BDD algorithms) should be developped in order to deal with parse trees of very large formulas.

Indeed, for the time being the good ratio between more involved syntactic algorithms (being under examination) and practical efficiency is not clearly worked out, all the more because previous works mainly focused upon model-theoretic methods. We claim nevertheless that the syntactic level can offer

much more information, with a reasonable cost.

The benchmarks we have conducted tend to support this claim, though these tests mostly concern random formulas. In order to get interesting benchmarks, we would like to test only "sensible" specifications (i.e. used in industrial contexts for instance), which seems to be a delicate task, being known that a large database of such specifications is not yet available.

Acknowledgement

The author would like to thank kindly Robert de Simone for fruitful discussions and collaboration.

References

[BCM+92] Jerry R. Burch, Edmund M. Clarke, Kenneth L. McMillan, David L. Dill, and L. J. Hwang. Symbolic model checking: 10e20 states and beyond. *Information and Computation*, 98(2):142–170, 1992.

[Bou97] Amar Bouali. Xeve: an esterel verication environment (version v1.3). Technical Report RT-0214, INRIA Sophia-Antipolis, December 1997.

[DGV99] Marco Daniele, Fausto Giunchiglia, and Moshe Y. Vardi. Improved automata generation for linear temporal logic. In *International Conference on Computer Aided Verification*, pages 249–260, 1999.

[DU] Documentation and User's. The objective caml system release 3.02.

[EH00] Kousha Etessami and Gerard J. Holzmann. Optimizing buchi automata. In *International Conference on Concurrency Theory*, pages 153–167, 2000.

[GO01] Paul Gastin and Denis Oddoux. Fast ltl to bchi automata translation, 2001.

[GPVW95] Rob Gerth, Doron Peled, Moshe Vardi, and Pierre Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *Protocol Specification Testing and Verification*, pages 3–18, Warsaw, Poland, 1995. Chapman & Hall.

[Hol97] Gerard J. Holzmann. The model checker SPIN. *Software Engineering*, 23(5):279–295, 1997.

[SB00] Fabio Somenzi and Roderick Bloem. Efficient bchi automata from ltl formulae. In *International Conference on Computer Aided Verification*, pages 53–65, 2000.

[VW94] Moshe Y. Vardi and Pierre Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, 1994.