LTL Translation Improvements in Spot

A. Duret-Lutz
LRDE / EPITA
14-16 rue Voltaire,
94276 Le Kremlin-Bicêtre Cedex
France
adl@lrde.epita.fr

Spot is a library of model-checking algorithms. This paper focuses on the module translating LTL formulæ into automata. We discuss improvements that have been implemented in the last four years, we show how Spot's translation competes on various benchmarks, and we give some insight into its implementation.

Model checking, Büchi automata, LTL, temporal logic, translation, simplifications, implementation

1. INTRODUCTION

One of the first steps of the automata-theoretic approach to model checking of linear-time properties (Vardi 1996, 2007) is to translate the property to verify into an automaton. This automaton is then synchronized with a model of the system in order to find executions that invalidate the property.

The Spot library (Duret-Lutz and Poitrenaud 2004) offers algorithms to realize the above automata-theoretic approach. A salient feature of Spot is its preference for using Transition-based Generalized Büchi Automata instead of the more commonly used Büchi Automata. Section 2 explains the difference.

This paper contains experience feedback on the implementation of a translator of linear-time temporal logic (LTL) formulæ into Büchi automata. Spot actually offers four translation procedures, and we shall only discuss the most efficient one, based on an algorithm by Couvreur (1999). We will often delve into technical details, because it is the various adjustments done before, inside, and after the translation algorithm that contribute to the quality of its output and to its efficiency. This paper accounts for 4 years of such development. Along the way, we report some errors that we have made and point some steps (like the degeneralization) that could probably be improved. We believe the insight we provide into the implementation of Spot should be helpful to anyone devising a translator.

We assume the reader is familiar with LTL (Clarke et al. 2000) and Binary Decision Diagrams (Bryant 1986), abbreviated as BDDs in the sequel.

This paper is organized as follows. Section 2 defines Transition-based Generalized Büchi Automata as opposed to Büchi Automata. Section 3 presents the core of the translation algorithm, with an emphasis on the optimizations that are enabled by the use of BDDs. Finally section 4 starts by explaining how *bad* the version of Spot praised by Rozier and Vardi (2007) was, and continues with a *potpourri* of improvements that have been achieved since.

Throughout the paper, the reader is invited to play with the on-line version of the translator available at `http://spot.lip6.fr/ltl2tgba.html`. It has options for many optimizations discussed herein.

2. TWO KINDS OF BÜCHI AUTOMATA

AP is a set of *atomic propositions*, i.e., propositional variables that may be true or false in the system. 2^{AP} denotes the set of minterms (or assignments) over AP, and $2^{2^{AP}}$ interpreted as the set of sums of minterms denotes the Boolean formulæ over AP.

Definition 1 *A Büchi automaton is a tuple* $\mathcal{B} = \langle AP, \mathcal{Q}, q^0, \mathcal{F}, \delta\rangle$ *where* AP *is a set of* atomic propositions*,* $\mathcal{Q}$ *is a finite set of states,* $q^0 \in \mathcal{Q}$ *is the initial state,* $\mathcal{F} \subseteq \mathcal{Q}$ *is a set of acceptance states, and* $\delta \subseteq \mathcal{Q} \times 2^{2^{AP}} \times \mathcal{Q}$ *is a transition relation in which each transition is labeled by a Boolean formula.*
An infinite word $c_0c_1c_2\ldots \in (2^{AP})^\omega$ *of assignments is* accepted *by* $\mathcal{B}$ *if there exists a run of* $\mathcal{A}$*, say* $(q^0, l_0, q_1)(q_1, l_1, q_2)(q_2, l_2, q_3)\ldots \in \delta^\omega$*, that recognizes the word (*$\forall i, c_i \in l_i$*) and that visits infinitely many acceptance states (*$\forall i \geq 0, \exists j \geq i, q_j \in \mathcal{F}$*).*

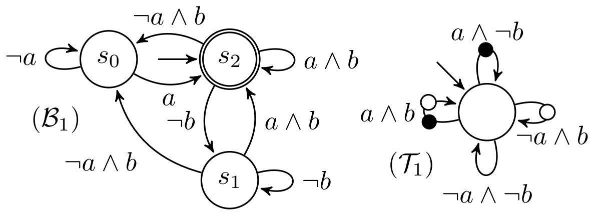


Figure 1: *Two automata regognizing the LTL formula* $\mathsf{G}\,\mathsf{F}\,a \wedge \mathsf{G}\,\mathsf{F}\,b$. $\mathcal{B}_1$*: Büchi automaton with a single acceptance state (double circle).* $\mathcal{T}_1$*: TGBA with* $\mathcal{F} = \{\bullet, \circ\}$.

Definition 2 *A Transition-based Generalized Büchi Automaton (TGBA) is a Büchi automaton in which multiple acceptance conditions are carried by the transitions. It can be defined as a tuple* $\mathcal{T} = \langle AP, \mathcal{Q}, q^0, \mathcal{F}, \delta \rangle$ *where* AP *is a set of* atomic propositions*,* $\mathcal{Q}$ *is a finite set of states,* $q^0 \in \mathcal{Q}$ *is the initial state,* $\mathcal{F} = \{f_1, f_2, \ldots, f_n\}$ *is a finite set of elements called* acceptance conditions*,* $\delta \subseteq \mathcal{Q} \times 2^{2^{AP}} \times 2^{\mathcal{F}} \times \mathcal{Q}$ *is a transition relation in which each transition is labeled by a Boolean formula and a set of acceptance conditions.*
An infinite word $c_0 c_1 c_2 \ldots \in (2^{AP})^\omega$ *of assignments is* accepted *by* $\mathcal{T}$ *if there exists a run of* $\mathcal{A}$*, say* $(q^0, l_0, F_0, q_1)(q_1, l_1, F_1, q_2)(q_2, l_2, F_2, q_3) \ldots \in \delta^\omega$*, that recognizes the word (*$\forall i, c_i \in l_i$*) and that visits each acceptance condition infinitely often (*$\forall f \in \mathcal{F}, \forall i \geq 0, \exists j \geq i,\ f \in F_j$*).*

Fig. 1 illustrates these definitions with two automata that recognize the LTL property: $\mathsf{G}\,\mathsf{F}\,a \wedge \mathsf{G}\,\mathsf{F}\,b$. The infinite sequence $\begin{smallmatrix} a\colon & 1\,0\,0\,1\,0\,0\,1\,0\,0\,\ldots \\ b\colon & 0\,0\,1\,0\,0\,1\,0\,0\,1\,\ldots \end{smallmatrix}$ will be accepted by $\mathcal{T}_1$ because it visits the top and right loops infinitely often, therefore all acceptance conditions are seen infinitely often. Similarly this sequence visits the only acceptance state of $\mathcal{B}_1$ infinitely often.

Spot is built around the concept of TGBA, and is able to perform the entire model-checking approach with these automata. However most other model-checking tools use Büchi automata. Spot can therefore *degeneralize* TGBAs into Büchi automata using an operation discussed in Section 4.2.2.

In the rest of the paper, we will often name the states of automata with the LTL formula they recognize. These extra annotations have no influence on the behavior of the automata.

3. OVERVIEW OF THE TRANSLATION

The algorithm of Couvreur (1999) for the translation of LTL automata into TGBA is based on a tableau method. Although the following explanations are self-contained, we refer the reader to Duret-Lutz and Poitrenaud (2004) for an illustration of this algorithm as a tableau that can be used to build generalized Büchi automata with state-based or transition-based acceptance conditions. Here we shall present the algorithm at a lower level to explain how the use of BDDs helps the translation.

To put this algorithm in context, the complete translation procedure to go from LTL to a Büchi Automaton can be presented as four steps:

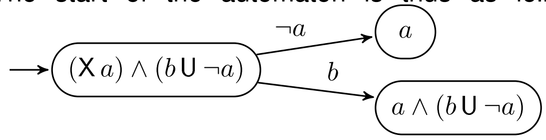
1. Simplify the LTL formula syntactically (we shall discuss this Section 4.2). E.g., rewrite $\mathsf{F}\,\mathsf{F}\,a$ (a 3-state automaton) into $\mathsf{F}\,a$ (2 states).
2. Translate the simplified formula into a TGBA using the algorithm presented in this section.
3. Simplify the TGBA, e.g., by removing redundant acceptance conditions and terminal strongly connected components that are not accepting.
4. Degeneralize the TGBA into a Büchi automaton, if desired, as discussed in section 4.2.2.

An final step could be to simplify the Büchi automaton, e.g., with simulation-based reductions (Etessami et al. 2001). We do not discuss this option because its implementation in Spot is a *work in progress* and the results are already good without it.

3.1. Basic Translation

If you omit BDDs, the procedure is simple enough to be performed by hand on a paper or blackboard[1]. The algorithm generates an automaton whose states corresponds to LTL formulæ. The initial state is the formula to translate. This formula is then rewritten as a sum of products where the only temporal operator allowed at the top level is X.

For instance if we were to translate $\Psi_1 = (\mathsf{X}\,a) \wedge (b\,\mathsf{U}\,\neg a)$ we would use the fact that $\varphi\,\mathsf{U}\,\psi = \psi \vee (\varphi \wedge \mathsf{X}(\varphi\,\mathsf{U}\,\psi))$ to rewrite Ψ_1 as $(\neg a \wedge \mathsf{X}\,a) \vee (b \wedge \mathsf{X}\,a \wedge \mathsf{X}(b\,\mathsf{U}\,\neg a))$. Reading this formula, it is clear that if your are in a state that must recognize Ψ_1, then you should either accept an assignment compatible with $\neg a$ and verify a at the next step, or accept an assignment compatible with b and then verify $a \wedge (b\,\mathsf{U}\,\neg a)$ at the next step. The start of the automaton is thus as follows:



The procedure should then be applied similarly on the new states. There is little subtlety that has to be taken into account when translating the $\varphi\,\mathsf{U}\,\psi$ operator: the formula ψ *must* occur eventually, it cannot be postponed infinitely. This is solved in the translation by expliciting a *promise to fulfil* ψ while rewriting the formula. The actual rewriting rule used

[1] Couvreur devised it while preparing a model-checking lecture.

$$
\begin{aligned}
r(\top) &= \top\\
r(\bot) &= \bot\\
r(p) &= \mathsf{Var}[p]\\
r(\neg p) &= \neg\mathsf{Var}[p]\\
r(f \vee g) &= r(f) \vee r(g)\\
r(f \wedge g) &= r(f) \wedge r(g)\\
r(\neg(f \vee g)) &= r(\neg f) \wedge r(\neg g)\\
r(\neg(f \wedge g)) &= r(\neg f) \vee r(\neg g)\\
r(\mathsf{X}\, f) &= \mathsf{Nxt}[f]\\
r(\neg\,\mathsf{X}\, f) &= \mathsf{Nxt}[\neg f]\\
r(f \,\mathsf{U}\, g) &= r(g) \vee (r(f) \wedge \mathsf{Nxt}[f \,\mathsf{U}\, g] \ \wedge\ \mathsf{P}[g])\\
r(\neg(f \,\mathsf{U}\, g)) &= r(\neg g) \wedge (r(\neg f) \vee \mathsf{Nxt}[\neg(f \,\mathsf{U}\, g)])
\end{aligned}
$$

Figure 2: *Rewriting rules to translate an LTL formula into a BDD. LTL patterns are in blue, BDD variables and operations in red.*

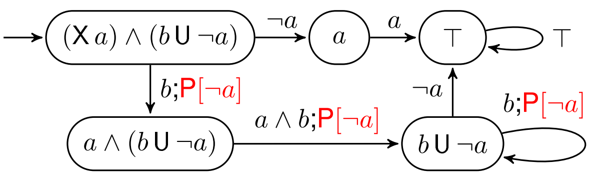


Figure 3: *Translation of* $(\mathsf{X}\, a) \wedge (b \,\mathsf{U}\, \neg a)$ *using promises.*

for U is: $\varphi \,\mathsf{U}\, \psi = \psi \vee (\varphi \wedge \mathsf{X}(\varphi \,\mathsf{U}\, \psi) \wedge \mathsf{P}\, \psi)$, with the operator P denoting an explicit promise.

All these formulæ can be simplified using classical Boolean rules like $(\alpha \wedge \beta) \vee \alpha = \alpha$ to kill some terms (even $\mathsf{X}\, \varphi$ or $\mathsf{P}\, \varphi$). This is where using BDD really helps. The core of the translation is the rewriting function $r(f)$ defined recursively as in Fig. 2. $\mathsf{Var}[p]$, $\mathsf{Nxt}[f]$, $\mathsf{P}[f]$, are BDD variables created as needed to represent respectively atomic propositions, $\mathsf{X}\, f$ formulæ, and promises.

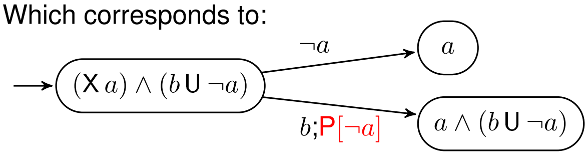
Applying r on our example, we obtain:

$$
\begin{aligned}
& r((\mathsf{X}\, a) \wedge (b \,\mathsf{U}\, \neg a)) = r(\mathsf{X}\, a) \wedge r(b \,\mathsf{U}\, \neg a)\\
&= \mathsf{Nxt}[a] \wedge (r(\neg a) \vee (\mathsf{P}[\neg a] \wedge r(b) \wedge \mathsf{Nxt}[b \,\mathsf{U}\, \neg a]))\\
&= \mathsf{Nxt}[a] \wedge (\neg\mathsf{Var}[a] \vee (\mathsf{P}[\neg a] \wedge \mathsf{Var}[b] \wedge \mathsf{Nxt}[b \,\mathsf{U}\, \neg a]))
\end{aligned}
$$

This BDD is then massaged into a sum of products:

$$
\begin{aligned}
&= (\neg\mathsf{Var}[a] \wedge \mathsf{Nxt}[a])\\
&\quad \vee\ (\mathsf{P}[\neg a] \wedge \mathsf{Var}[b] \wedge \mathsf{Nxt}[a] \wedge \mathsf{Nxt}[b \,\mathsf{U}\, \neg a])
\end{aligned}
$$

Which corresponds to:



There are several ways to turn a BDD into a sum of products, but because each term of the sum corresponds to a transition in the automaton, redundant terms should be avoided. We use an algorithm from Minato (1992) to that effect.

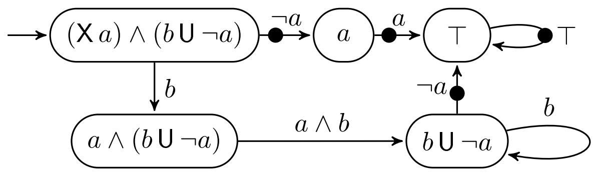


Figure 4: *Translation of* $(\mathsf{X}\, a) \wedge (b \,\mathsf{U}\, \neg a)$ *as a TGBA.*

ltl_to_tgba_fm(f)
  $todo \leftarrow \{f\}$; $all\_acc \leftarrow \emptyset$
  $a \leftarrow$**new** automaton; a.set_initial_state(f)
  while ($todo \neq \emptyset$)
    $here \leftarrow todo$.remove_one()
    forall i **in** $prime\_implicants\_of(r(here))$
      Put i as $\bigwedge_{v \in V} \mathsf{Var}[v] \wedge \bigwedge_{v \in V'} \neg\mathsf{Var}[v] \wedge \bigwedge_{a \in A} \mathsf{P}[a] \wedge \bigwedge_{n \in N} \mathsf{Nxt}[n]$
      $dest \leftarrow \bigwedge_{n \in N} n$
      if $\neg a$.has_state($dest$)
        $todo$.insert($dest$)
        a.add_transition(src: $here$, dst: $dest$,
          cond: $\bigwedge_{v \in V} v \wedge \bigwedge_{v \in V'} \neg v$, promises: A)
        $all\_acc \leftarrow all\_acc \cup A$
  forall t **in** a.transitions()
    t.acceptance_conditions$\leftarrow all_acc \setminus t$.promises
  return a

Figure 5: *Pseudo-code of the algorithm of Couvreur (1999) to translate an LTL formula* f *into a TGBA. The function* $r(here)$ *is defined on Fig. 2.*

The complete translation is shown on Fig. 3. This automaton is still not a TGBA because it uses promises instead of Büchi acceptance conditions. To guarantee that a promise holds, the accepted runs of the automaton should never make promises continuously: in other words for each promise $\mathsf{P}\, \varphi$, accepted runs should visit infinitely many transitions that do not make such a promise.[2] This can be encoded as a TGBA by labelling all transitions that do not make promise $\mathsf{P}\, \varphi$ by an acceptance condition associated to φ.[3] There will be as many acceptance conditions as promises. Fig. 4 shows the final TGBA.

The pseudo-code for the complete translation algorithm is shown on Fig. 5.

At this point it should be clear that the use of BDDs simplifies every Boolean formulæ that label transitions. For instance we cannot have a transition labelled by $b \wedge a \wedge \neg b$ because such a conjunction would be simplified by the BDD representation.[4] Similarly the conversion of the BDD into an irredundant sum

[2] Promises should not be mistaken for co-Büchi acceptance conditions. A co-Büchi acceptance condition $\mathcal{F}$ accepts runs that stay in $\mathcal{F}$ continuously; conversely a promise accepts runs that do not make the promise continuously.
[3] Spot use the syntax Acc[f] to display an acceptance conditions associated to the formula f. This syntax can be seen when using our on-line LTL translator. In this paper we simply display these acceptance conditions as dots of various colors (○, ●, . . .).
[4] Some translators will output similar labels: see footnote 7.

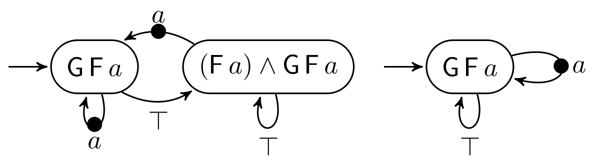


Figure 6: *Two translations of* $\mathsf{G}\,\mathsf{F}\,a$*. Since* $r(\mathsf{G}\,\mathsf{F}\,a) = r((\mathsf{F}\,a) \wedge \mathsf{G}\,\mathsf{F}\,a)$ *the two states of the first automaton can be merged, yielding the second automaton.*

of products helps to reduce the number of outgoing arcs of each node. As an illustration, consider the formula $\neg((a\,\mathsf{U}\,b) \vee (b\,\mathsf{U}\,c))$. Blindly applying the tableau rules listed by Duret-Lutz and Poitrenaud (2004) will produce 4 successors corresponding to $(\neg a) \wedge (\neg b) \wedge (\neg c)$, $(\neg a) \wedge (\neg b) \wedge (\neg c) \wedge \mathsf{X}\,\neg(b\,\mathsf{U}\,c)$, $(\neg b) \wedge (\neg c) \wedge (\mathsf{X}\,\neg(a\,\mathsf{U}\,b))$, and $(\neg b) \wedge (\neg c) \wedge (\mathsf{X}\,\neg(a\,\mathsf{U}\,b)) \wedge \mathsf{X}\,\neg(b\,\mathsf{U}\,c)$ with some obvious redundancy. On the other hand the corresponding BDD rewriting will be simplified to $\neg\mathsf{Var}[b] \wedge \neg\mathsf{Var}[c] \wedge (\neg\mathsf{Var}[a] \vee \mathsf{Nxt}[\neg(a\,\mathsf{U}\,b)])$, giving only the two successors $(\neg a) \wedge (\neg b) \wedge (\neg c)$ and $(\neg b) \wedge (\neg c) \wedge (\mathsf{X}\,\neg(a\,\mathsf{U}\,b))$.

3.2. Using r to Identify States

A powerful BDD-based optimization is to use the r function to identify some equivalent formulæ. Because BDDs have a unique representation, we have that $r(\varphi) = r(\psi) \implies \varphi = \psi$. The converse does not hold because two equivalent subformulæ prefixed with X might be represented by different $\mathsf{Nxt}[]$ variables. Graphically, $r(\varphi)$ encodes the outgoing transitions (labels, promises, and destinations) of the state ψ, so $r(\varphi) = r(\psi)$ means that the states φ and ψ have the same successors and can be merged.

Such a reduction occurs when translating $\mathsf{G}\,\mathsf{F}\,a$:

$$r(\mathsf{G}\,\mathsf{F}\,a) = ((\mathsf{Nxt}[\mathsf{F}\,a] \wedge \mathsf{P}[a]) \vee \mathsf{Var}[a]) \wedge \mathsf{Nxt}[\mathsf{G}\,\mathsf{F}\,a]$$

$$r((\mathsf{F}\,a) \wedge \mathsf{G}\,\mathsf{F}\,a) = ((\mathsf{Nxt}[\mathsf{F}\,a] \wedge \mathsf{P}[a]) \vee \mathsf{Var}[a]) \wedge \mathsf{Nxt}[\mathsf{G}\,\mathsf{F}\,a]$$

The result of $r(\mathsf{G}\,\mathsf{F}\,a)$ implies that $\mathsf{G}\,\mathsf{F}\,a$ should have two successors $\mathsf{G}\,\mathsf{F}\,a$ and $(\mathsf{F}\,a) \wedge \mathsf{G}\,\mathsf{F}\,a$ as shown in the first automaton of Fig. 6. However $r((\mathsf{F}\,a) \wedge \mathsf{G}\,\mathsf{F}\,a) = r(\mathsf{G}\,\mathsf{F}\,a)$ so these states can be merged.

One way to implement this reduction automatically is to index the states of the automata by the BDD $r(\varphi)$ instead of by the LTL formula φ (the pseudo-code from Fig. 5 does *not* perform this reduction).

3.3. Better Determinism

The determinism of the automata from Fig. 6 can be improved using a trick based on the BDD representation of states. Instead of converting the equation $r(\mathsf{G}\,\mathsf{F}\,a) = ((\mathsf{Nxt}[\mathsf{F}\,a] \wedge \mathsf{P}[a]) \vee \mathsf{Var}[a]) \wedge \mathsf{Nxt}[\mathsf{G}\,\mathsf{F}\,a]$ into a sum of products to discover the labels and destination, we can instead fix one label to discover its destination(s).

Where shall we go if we read a?

$$r(\mathsf{G}\,\mathsf{F}\,a) \wedge \mathsf{Var}[a] = \mathsf{Var}[a] \wedge \mathsf{Nxt}[\mathsf{G}\,\mathsf{F}\,a]$$

Where shall we go if we read $\neg a$?

$$r(\mathsf{G}\,\mathsf{F}\,a) \wedge \neg\mathsf{Var}[a] = \neg\mathsf{Var}[a] \wedge \mathsf{Nxt}[\mathsf{F}\,a] \wedge \mathsf{P}[a] \wedge \mathsf{Nxt}[\mathsf{G}\,\mathsf{F}\,a]$$

These equations show that all instances of $\top$ in Fig. 6 can be replaced by $\neg a$, yielding two deterministic automata.

In an automaton over n atomic propositions ($\mathsf{Var}[a]$, $\mathsf{Var}[b]$,...), there are 2^n labels to consider. However the structure of the BDD encoding the formula helps to ignore useless labels; and in real-world formulæ n is usually small enough so that the slowdown incurred by this enumeration is not perceptible.

An automaton constructed this way is usually *more* deterministic, but it is not necessarily a deterministic automaton. The result of $r(\varphi) \wedge A$ for some A could have a disjunction, i.e., multiple destinations.

In an experiment we translated 96 LTL formulæ taken from the literature with and without this optimization. The resulting automata were then synchronized with random state spaces to measure the effect of the improved determinism. With this technique the number of transitions in the product was reduced by 40%, and the number of states by only 0.33%.

4. FROM SPOT-0.4 TO SPOT-0.7.1

We now review how this translation has been improved over the last four years.

4.1. Better Data Structures

Rozier and Vardi (2007) compared 9 LTL translators, on various families of LTL formulæ.

The first family of formulæ they experimented is scalable. For a given n they generated[5] an LTL formula C_n that matches an infinite sequence of bits in which all the values of a n-bit counter have been concatenated. E.g., $C_3 = ((a \wedge (\mathsf{G}(a \rightarrow (\mathsf{X}(\neg a \wedge \mathsf{X}(\neg a \wedge \mathsf{X}\,a)))))) \wedge ((\neg b) \wedge \mathsf{X}(\neg b \wedge \mathsf{X}\,\neg b)) \wedge (\mathsf{G}((a \wedge \neg b) \rightarrow (\mathsf{X}((\mathsf{X}\,\mathsf{X}\,b) \wedge (((\neg a) \wedge (b \rightarrow \mathsf{X}\,\mathsf{X}\,\mathsf{X}\,b) \wedge ((\neg b) \rightarrow (\mathsf{X}\,\mathsf{X}\,\mathsf{X}\,\neg b))) \,\mathsf{U}\, a))))) \wedge (\mathsf{G}((a \wedge b) \rightarrow (\mathsf{X}((\mathsf{X}\,\mathsf{X}\,\neg b) \wedge ((b \wedge (\neg a) \wedge \mathsf{X}\,\mathsf{X}\,\mathsf{X}\,\neg b) \,\mathsf{U}\, (a \vee ((\neg a) \wedge (\neg b) \wedge (\mathsf{X}((\mathsf{X}\,\mathsf{X}\,b) \wedge (((\neg a) \wedge (b \rightarrow \mathsf{X}\,\mathsf{X}\,\mathsf{X}\,b) \wedge ((\neg b) \rightarrow \mathsf{X}\,\mathsf{X}\,\mathsf{X}\,\neg b)) \,\mathsf{U}\, a)))))))))))$. Such a formula will match a sequence consisting of

$$\begin{array}{l} a\colon 1\;0\;0\;1\;0\;0\;1\;0\;0\;1\;0\;0\;1\;0\;0\;1\;0\;0\;1\;0\;0\;1\;0\;0\;\ldots \\ b\colon 0\;0\;0\;1\;0\;0\;0\;1\;0\;1\;1\;0\;0\;0\;1\;1\;0\;1\;0\;1\;1\;1\;1\;1\;\ldots \end{array}$$

repeated infinitely. Variable a beats the start of each value, while variable b iterates over the 3 bits of each value from least to most significant bit $(000, 100, 010, \ldots)$.

From this description it should be clear that the smallest automaton that can recognize C_n is a

[5]Scripts to generate these formulæ can be found at `http://ti.arc.nasa.gov/m/profile/kyrozier/benchmarking_scripts/benchmarking_scripts.html`

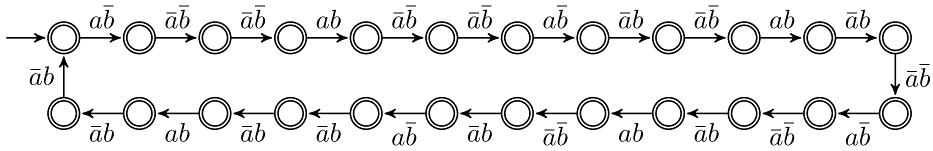


Figure 7: *A Büchi automaton that recognizes* C_3.

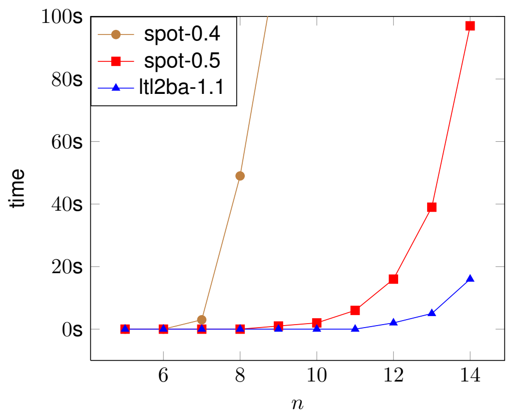


Figure 8: *Runtime of two Spot versions on LTL counter formulæ. ltl2ba is shown for comparison.*

deterministic loop with $n2^n$ states and as many transitions. Fig. 7 shows this automaton for C_3. Any translator that constructs such an automaton explicitly will have a runtime that is worse than exponential in n.

Rozier and Vardi (2007) show graphs comparing how fast the runtime of each translator grows: few tools passed $n = 8$. Although the Spot runtime was low compared to other tools (even symbolic) they stopped using Spot after $n = 9$ because it generated an incorrect automaton for C_{10}.

Investigating this bug[6] and playing with C_n revealed that Spot's implementation of LTL formulæ and its implementation of automata were suboptimal. The effect of the subsequent fixes can been seen on figure 8: spot-0.4 is the version used by Rozier and Vardi, while spot-0.5 is an improved version that is 5 times faster: only the formulæ and automata implementation were changed, the translation algorithm is the same. ltl2ba-1.1 (Gastin and Oddoux 2001) is shown for comparison, and other tools are usually off the scale: for instance spin-6.1.0 will take more that 11 hours to translate the formula C_1 describing a 2-state automaton[7].

In Spot, LTL formulæ are represented as "abstract syntax DAGs", i.e., identical subformulæ are represented only once, as in Fig. 9. For commutative operators, the operands are systematically reordered[8] so that we can easily detect that $a \wedge \mathsf{X}(b)$ is equal to $\mathsf{X}(b) \wedge a$. Sharing of subformulæ also work across multiple formulæ. The uniqueness of each subformula helps to speed up algorithms that may use a cache as they traverse formulæ. However this setup requires that each subformula in the pool be reference counted. The first issue in spot-0.4 was how this reference counting worked: the counter of each node recorded the number of each subformula using this node. This meant that every time a formula was cloned or destroyed, we had to recurse its entire DAG to increment or decrement the counters of each node... "27%" of the time of the translation of C_8 was spent dealing with counters. The obvious fix was to count only the number of direct parents of a node, and this code ran no more than "0.01%" of the time.

The second problem was in the interface used to construct Büchi automata as explicit graphs. The definition of a Büchi automaton does not specify how to represent the elements of Q, i.e., what is the identity of a state. Is it an integer? Is it a name? When reading an automaton from a file written by hand, we want to allow any string. When implementing an LTL translation such as the one described in section 3, it is tempting to label states with the LTL formula they represent: it helps reading the automaton. The states of explicit graphs in spot-0.4 were referenced by strings to accommodate all these needs. The translation algorithm was therefore wasting a lot of time just converting LTL formulæ into strings to designate states. A new kind of graph with states labeled by pointers to LTL formula objects yielded the most part of the improvement shown on Fig. 8.

4.2. LTL Simplification Improvements and Better Degeneralization

After the release of Spot 0.5, we were contacted by Rüdiger Ehlers who was trying to use ltl2ba and Spot with various options to translate an LTL formulæ and keep only the smallest automata (in term of states) produced by any tool. His goal was to later minimize these automata *even more* using a

[6] It was a typo in the comparison function used to distinguish two LTL formulæ that have the same hash value: a constant '1' (one) was used instead of the variable 'l' (lowercase L), but the two characters were indistinguishable on screen! C_{10} was the first formula with enough sub-formulæ to trigger a hash conflict.

[7] The automaton built by spin-6.1.0 for C_1 actually has 33 states and 447 transitions, many of them using unsatisfiable guards such as "`((!b) && (a) && (b))`".

[8] Each subformula gets a serial number the first time it is created, so this gives a handy key to order them in a way that is more deterministic than ordering by their address.

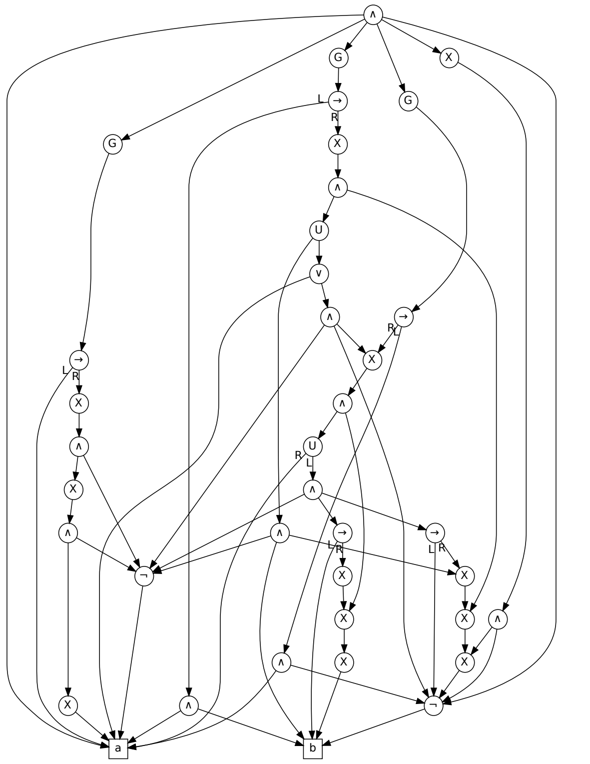


Figure 9: *An abstract syntax DAG for the formula C_3. The letters **L** and **R** are used to distinguish left and right operands of non-commutative binary operators.*

worse \ better	ltl2ba-1.1	spot-0.5 (1)	spot-0.5 (2)	spot-0.6 (1)	spot-0.6 (2)
ltl2ba-1.1	0	23	28	58	61
spot-0.5 (1)	80	0	6	104	107
spot-0.5 (2)	76	0	0	101	102
spot-0.6 (1)	2	0	4	0	4
spot-0.6 (2)	1	0	0	0	0

Table 1: *Comparing ltl2ba and Spot on 184 LTL formulæ. The value on line i and column j shows how many time automaton produced by translator $\#i$ was strictly bigger than the automaton produced by translator $\#j$.*

technique that could easily take hours (Ehlers and Finkbeiner 2010), so it made sense to try a few tools to find the smallest candidate first.

The benchmark consists in 92 LTL formulæ:

- 55 formulæ from Dwyer et al. (1998),
- 25 formulæ from Somenzi and Bloem (2000) — their paper shows 27 formulæ but two of them were ignored because they are already the negations of other formulæ in the list,
- 12 formulæ from Etessami and Holzmann (2000).

With their negations this makes a total of 184 formulæ. Table 1 shows how much time each translator was better than the other on these 184 formulæ. For instance comparing "spot-0.5 (1)" with "ltl2ba-1.1" we can see ltl2ba-1.1 produced automata strictly smaller than spot-0.5 for 80 formulæ, while spot-0.5 was strictly better in 23 cases.

The numbers (1) and (2) refer to different reductions performed on the LTL formulæ before translation.

In case (1), each formula is reduced using rules taken from Somenzi and Bloem (2000); Etessami and Holzmann (2000) and Bloem (2001). These rules include unconditional rewritings like "$\mathsf{F}\,\mathsf{X}\,\varphi = \mathsf{X}\,\mathsf{F}\,\varphi$", conditional rewriting based on syntactic implications[9] like "if $\varphi \Rightarrow \psi$ then $\varphi \,\mathsf{U}\, \psi = \psi$", and rewritings based on the concept pure eventualities and purely universal formulæ like "if ψ is a pure eventuality then $\varphi \,\mathsf{U}\, \psi = \psi$"[10]. ltl2ba-1.1 includes similar rules, except that pure eventualities and purely universal formulæ are handled with a less general set of rewritings.[11]

In case (2), a stronger set of rules based on language containment are used to replace the syntactic implications checks. In this setup, checking that $\varphi \Rightarrow \psi$ involves translating φ and $\neg\psi$ into automata, and checking whether their product is empty: $\mathscr{L}(A_\varphi \otimes A_{\neg\psi}) = \emptyset$.[12] These operations have to be performed on many subformulæ of the formula to simplify so this is a costly simplification. But have found that the runtime is usually acceptable as the automata involved when translating formulæ used in verification are usually small. The initial set of rewriting rules based on language containment comes from Tauriainen (2006, section 5.3), but we extended these as in Table 2 so they would catch more patterns to reduce. The simplification described for (1) are still performed, but language-containment rules are able to catch a few more simplifications that were not detected syntactically.

A typical formula that rules (1) are not able to simplify is $\varphi_1 = ((\mathsf{X}\,q) \wedge r)\,\mathsf{R}\,\mathsf{X}(((s\,\mathsf{U}\,p)\,\mathsf{R}\,r)\,\mathsf{U}(s\,\mathsf{R}\,r))$. The extended language containment rules from Table 2 reduces φ_1 to $\varphi_2 = ((\mathsf{X}\,q) \wedge r)\,\mathsf{R}\,\mathsf{X}(s\,\mathsf{R}\,r)$ because $((s\,\mathsf{U}\,p)\,\mathsf{R}\,r)\,\mathsf{U}(s\,\mathsf{R}\,r)$ implies $(s\,\mathsf{R}\,r)$. The reduction this entails on the automaton is welcome: φ_1 is translated by Spot into a Büchi automaton with 25 states, while φ_2 needs only 5 states.

[9] This means that the condition $\varphi \Rightarrow \psi$ is tested syntactically by a recursion on the two abstract syntax DAGs.

[10] For instance $\mathsf{F}\,b$ is a pure eventuality, so $a\,\mathsf{U}\,\mathsf{F}\,b = \mathsf{F}\,b$.

[11] For instance ltl2ba-1.1 can reduce $a\,\mathsf{U}\,\mathsf{F}\,b$ to $\mathsf{F}\,b$, but it cannot reduce $a\,\mathsf{U}\,\mathsf{X}\,\mathsf{F}\,b$ to $\mathsf{X}\,\mathsf{F}\,b$ although $\mathsf{X}\,\mathsf{F}\,b$ is a pure eventuality.

[12] The reason why we do not simply check that $\mathscr{L}(A_{\varphi\wedge\neg\psi}) = \emptyset$ is so we can cache the automata generated A_φ and $A_{\neg\psi}$ and use them in when checking other conditions involving the same subformulæ.

pattern	condition to test	simplification
$\varphi \vee \psi$	$\mathscr{L}(A_\varphi \otimes A_{\neg\psi}) = \emptyset$	ψ
	$\mathscr{L}(A_{\neg\varphi} \otimes A_{\psi}) = \emptyset$	φ
	$\mathscr{L}(A_{\neg\varphi} \otimes A_{\neg\psi}) = \emptyset$	$\top$
$\varphi \wedge \psi$	$\mathscr{L}(A_\varphi \otimes A_{\neg\psi}) = \emptyset$	φ
	$\mathscr{L}(A_{\neg\varphi} \otimes A_{\psi}) = \emptyset$	ψ
	$\mathscr{L}(A_{\varphi} \otimes A_{\psi}) = \emptyset$	$\top$
$\varphi \mathsf{U} \psi$	$\mathscr{L}(A_{\varphi \mathsf{U} \psi} \otimes A_{\neg\psi}) = \emptyset$	ψ
	$\mathscr{L}(A_{\neg\varphi} \otimes A_{\neg\psi}) = \emptyset$	$\mathsf{F}\,\psi$
$\varphi \mathsf{R} \psi$	$\mathscr{L}(A_\varphi \otimes A_{\neg\psi}) = \emptyset$	ψ
	$\mathscr{L}(A_{\varphi} \otimes A_{\psi}) = \emptyset$	$\mathsf{G}\,\psi$
$\mathsf{X}\,\varphi$	$\big(\mathscr{L}(A_\varphi \otimes A_{\neg \mathsf{X}\,\varphi}) = \emptyset$ $\wedge \mathscr{L}(A_{\neg\varphi} \otimes A_{\mathsf{X}\,\varphi}) = \emptyset\big)$	φ

Table 2: *Simplification rules based on language containment.*

After translating the simplified formula into a generalized Büchi automaton, both Spot and ltl2ba perform a pass of automaton simplification based on the elimination of maximal strongly connected components that are terminal and non-accepting. The degeneralization of this automaton into a Büchi automaton follows.

Looking at the formulæ for which Spot fared worse than ltl2ba in Table 1 we were able to make three improvements, included in spot-0.6. The first two changes were enough to bring Spot up to the point were only one formula was translated worse than ltl2ba. The third change improved the number of case where Spot was producing smaller automata.

4.2.1. A Harmful Rewriting Rule

The first correction is quite simple. In the set of rewriting rules chosen to reduce the size of the automaton produced by the Wring tool, Somenzi and Bloem (2000) give this one (applied from left to right):

$$\mathsf{F}(\varphi \wedge \mathsf{G}\,\mathsf{F}\,\psi) = (\mathsf{F}\,\varphi) \wedge \mathsf{G}\,\mathsf{F}\,\psi$$

It might be true that Wring performs better on the right formula, but this is generally wrong for Spot. This rule had to be disabled. Intuitively, this rule is dubious because $\mathsf{F}(\varphi \wedge \mathsf{G}\,\mathsf{F}(\psi))$ appears less complex to translate. Translating $\mathsf{F}\,\Phi$ is just a matter of creating an initial state that accepts any letter for a finite number of step, and non-deterministically jumps into a state that will recognize Φ when a letter matching the beginning of Φ is found. Translating a formula such as $(\mathsf{F}\,\varphi) \wedge \mathsf{G}\,\mathsf{F}\,\psi$ is harder because in the initial state you have four choices to consider: either the input can be the start of φ, or it is the start of ψ, or it is both, or it is none. When φ and ψ are atomic propositions as in Fig. 10, these four cases can be reduced to three. It turns out that on the automaton $\mathcal{A}_2$ from Fig. 10 the states $(\mathsf{F}\,a) \wedge \mathsf{G}\,\mathsf{F}\,b$ and

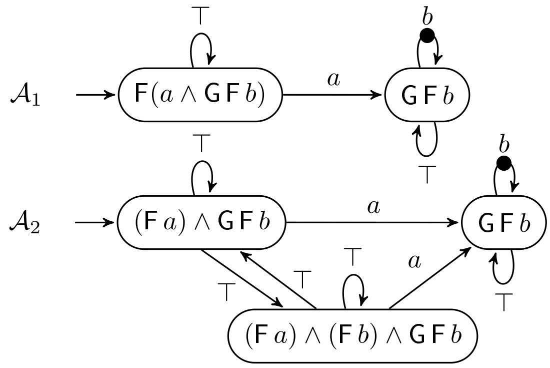


Figure 10: *Paper-and-pen translations into TGBA of* $\mathsf{F}(a \wedge \mathsf{G}\,\mathsf{F}\,b)$ *and* $(\mathsf{F}\,a) \wedge \mathsf{G}\,\mathsf{F}\,b$.

$(\mathsf{F}\,a) \wedge (\mathsf{F}\,b) \wedge \mathsf{G}\,\mathsf{F}\,b$ have exactly the same outgoing transitions so they can be merged. Thank to the BDD identification discussed in section 3.2, Spot will actually output an automaton similar to $\mathcal{A}_1$ for both formulæ $\mathsf{F}(a \wedge \mathsf{G}\,\mathsf{F}\,b)$ or $(\mathsf{F}\,a) \wedge \mathsf{G}\,\mathsf{F}\,b$. This is not the case when φ and ψ are more complex.

This rewriting rule also prevented other useful rules to apply. E.g., spot-0.5 would rewrite the formula $\mathsf{F}(\varphi_1 \wedge \mathsf{G}\,\mathsf{F}\,\psi_1) \vee \mathsf{F}(\varphi_2 \wedge \mathsf{G}\,\mathsf{F}\,\psi_2)$ as $((\mathsf{F}\,\varphi_1) \wedge \mathsf{G}\,\mathsf{F}\,\psi_1) \vee (\mathsf{F}\,\varphi_2) \wedge \mathsf{G}\,\mathsf{F}\,\psi_2)$ missing the opportunity to apply the rule $\mathsf{F}(\Psi_1) \vee \mathsf{F}(\Psi_2) = \mathsf{F}(\Psi_1 \vee \Psi_2)$. Spot-0.6 rewrites this formula as $\mathsf{F}((\varphi_1 \wedge \mathsf{G}\,\mathsf{F}\,\psi_1) \vee (\varphi_2 \wedge \mathsf{G}\,\mathsf{F}\,\psi_2))$ which is easier to translate for similar reasons.

In spot-0.6 (and also the current spot-0.7.1) this harmful rewriting rule is simply disabled, but from the above discussion it sounds like it would be better to apply it in the opposite direction, trying to gather as much terms as possible after the F. We are currently overhauling the entire rewriting module in order to improve it, and plan to make a more systematic study of each rule we apply. Applying a rewriting rule *because it has been used in another tool* is not a good justification: the rule might be favorable to the other tool but not yours.

4.2.2. Better Degeneralization

A degeneralization algorithm takes a generalized automaton with n states and m acceptance conditions, and produces a Büchi automaton with a single acceptance condition and at most $n(m+1)$ states. The classical algorithm used to transform Generalized Büchi Automata into Büchi automata (Clarke et al. 2000, section 9.2.2) can be adapted to transform TGBA into Büchi automata (Giannakopoulou and Lerda 2002; Gastin and Oddoux 2001) as follows.

If $\mathcal{T} = \langle AP, \mathcal{Q}, q_0, \mathcal{F}, \delta\rangle$ is a TGBA with m acceptance conditions $\mathcal{F} = \{f_1, f_2, \ldots, f_m\}$, then an equivalent Büchi automaton $\mathcal{T} = \langle AP, \mathcal{Q}', q_0', \mathcal{F}', \delta'\rangle$ can be constructed as follows:

- $\mathcal{Q}' = \mathcal{Q} \times \{0, \ldots, m\}$ the original automaton is cloned in $m+1$ levels,
- $\mathcal{F}' = \mathcal{Q} \times \{m\}$ states from the last level are accepting,
- $\delta' = \{((s,j), l, (d, L_j(F))) \mid (s,l,F,d) \in \delta\}$ where $L_j(F) = \begin{cases} 0 & \text{if } j = m \\ j+1 & \text{if } f_{j+1} \in F \\ j & \text{otherwise} \end{cases}$ i.e., for each level $j < m$ the outgoing transitions that carry f_{j+1} are redirected to the next level and all outgoing transitions from the last level are redirected to the first one.
- $q_0' = q_0 \times \{0\}$ the initial state is on the first level (but any other level would also be correct).

This setup guarantees that any accepting path in the degeneralized automaton will correspond to an infinite path that sees all acceptance conditions infinitely often in the original automaton. The classical optimization is to "jump levels", i.e., when a transition from level $i < m$ carries acceptance conditions f_{i+1}, f_{i+2}, and f_{i+3}, it can be redirected to the level $i+3$. This corresponds to the following redefinition of $L_j(F) =$
$\max\{n \in \{j, \ldots, m\} \mid \forall k \in \{j+1, \ldots, n\}, f_k \in F\}$ if $j < m$,
$\max\{n \in \{0, \ldots, m\} \mid \forall k \in \{1, \ldots, n\}, f_k \in \mathsf{F}\}$ if $j = m$

The automaton $\mathcal{B}_1$ from Fig. 1 was degeneralized from $\mathcal{T}_1$ with this definition, in the order $f_1 = \circ$, $f_2 = \bullet$, and setting the initial state in the last level.

This degeneralization procedure offers $m!$ possible ways to order the acceptance conditions, and there are $m+1$ possible levels on which the initial state can be located. Changing these parameter might make some states from $\mathcal{Q} \times \{0, \ldots, m\}$ unreachable, and can thus reduce the automaton. For one TGBA, we therefore have $m!(m+1)$ possible degeneralizations using only this definition.

In Spot, the acceptance conditions labeling each transitions are stored as BDDs[13] and the order used by the degeneralization algorithm is related to the order in which the corresponding BDD variables were declared. This order is in turn related to the order in which the LTL formula was recursively traversed by the translation algorithm. In ltl2ba, this order is that of the states of the alternating automaton is that constructed recursively from the LTL formula. So in both cases, the order used is tied to the syntax tree/DAG of the LTL formula. It so turns out that the order used in spot-0.5 was usually the reverse of the order used in ltl2ba, and was unfavorable. *Simply reversing that order, is responsible the largest part of the improvements observed between spot-0.5 and spot-0.6 on Table 1.* This reversed order corresponds to expecting the acceptance conditions associated to a formula ϕ before expecting the acceptance condition associated to subformulæ of ϕ. For instance when translating the formula $\mathsf{G}\,\mathsf{F}(a \wedge \mathsf{G}\,\mathsf{F}(b \wedge \mathsf{G}\,\mathsf{F}\,c))$ there will be acceptance conditions associated to $a \wedge \mathsf{G}\,\mathsf{F}(b \wedge \mathsf{G}\,\mathsf{F}\,c)$, $b \wedge \mathsf{G}\,\mathsf{F}\,c$ and c and they should be expected in this order during the degeneralization.

We are currently investigating how choosing different orders may improve the degeneralization and it seems that the above heuristic is usually better. The choice of the level for the initial state is another parameter that we have yet to explore.

Oddoux (2003, section 6.1.2) mentions another kind of degeralization in which the acceptance conditions can be taken in any order and where each state of degeneralized automaton has to retain the set of all acceptance conditions that are waited for. This can potentially multiply the size of the original automaton with 2^m if m acceptance conditions are used. But this might be worth a try when m is very small.

Another optimization that was included in spot-0.6 to improve its degeneralization is the "*pulling of acceptance conditions*". When all the outgoing transition of a state s have a set Y of acceptance conditions in common, this set can be added to the acceptance conditions of all the incoming transitions. This is correct because if a run of traverses s it will necessarily see all acceptance conditions from Y; it makes no difference if its sees them twice.

Finally the only formula of this benchmark for which ltl2ba-1.1 produces an automata with less states than spot-0.6 in Table 1 is $\neg\,\mathsf{G}(p \rightarrow (q\,\mathsf{U}(\mathsf{G}\,r \vee \mathsf{G}\,s)))$. ltl2ba-1.1 gives a Büchi automaton with 10 states and 336 transitions, while spot-0.6 produces 11 states and 260 transitions.[14] In this extraordinary case, disabling the SCC-based simplifications will actually help the degeneralization algorithm, and Spot will produce an automaton with 10 states and 244 transitions.

4.2.3. More LTL Operators: W *and* M

The changes described in sections 4.2.1 and 4.2.2 were enough to let Spot-0.6 (2) produce automata that were smaller that ltl2ba's in 50 cases. 11 more cases were won by adding support for the W and M operators, even if they are not used in the formulæ...

[13]In some algorithms, this enables union or intersection of sets of BDDs. But the real justification for this interface is that we also have some symbolic representations of automata where the transition relation is stored as a BDD from which we can extract those acceptance conditions using BDD manipulations.

[14]When counting transitions in an automaton over the atomic propositions $\{p, q, r, s\}$ an arc labeled by p is counted for 8 transitions because there are 8 different configurations of the atomic propositions ($p\bar{q}\bar{r}\bar{s}, pq\bar{r}\bar{s}, p\bar{q}r\bar{s}, \ldots$) that satisfy formula p.

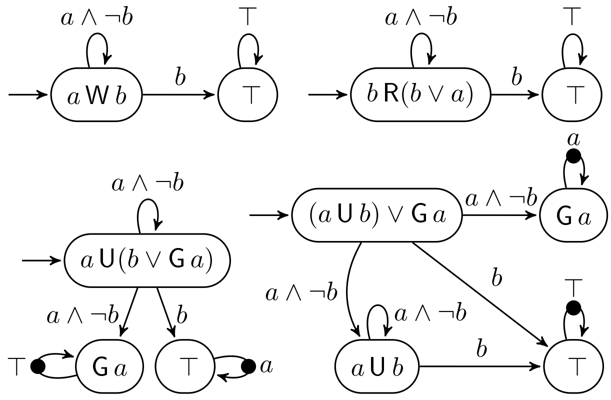


***Figure 11:** Four formulæ equivalent to $a \,\mathsf{W}\, b$ and their corresponding automata. When the W and R operators are not available, $a \,\mathsf{U}(b \vee \mathsf{G}\, a)$ is easier to translate.*

$a \,\mathsf{U}\, b$ is the *strong until* operator: a has to hold until b does, and b must hold eventually. $a \,\mathsf{W}\, b$ is the *weak until* operator: a should hold until b does, and if b never occur, then a should always be true.

Conversely, $a \,\mathsf{R}\, b$ is the *weak release* operator: b has to hold either infinitely, or until $a \wedge b$ hold together. $a \,\mathsf{M}\, b$ is the *strong release operator*: a has to hold until $a \wedge b$ hold, and the latter is will occur eventually.

These operators do not add expressive power: when using tools that do not support them, $a \,\mathsf{W}\, b$ is usually rewritten $(a \,\mathsf{U}\, b) \vee \mathsf{G}\, a$ or $a \,\mathsf{U}(b \vee \mathsf{G}\, a)$. Even if it is less intuitive, the second rewriting is actually a better choice, because it translates into a smaller automaton (Fig. 11). An even better choise, if the R operator is available, is to write $b \,\mathsf{R}(a \vee b)$. Similarly, $a \,\mathsf{M}\, b$ can be rewritten in many ways: $a \,\mathsf{M}\, b = (a \,\mathsf{R}\, b) \wedge (\mathsf{F}\, a) = a \,\mathsf{R}(b \wedge \mathsf{F}\, a) = b \,\mathsf{U}(a \wedge b)$.

The translation rule for W is the same as that for U except that no promise is made (hence no acceptance condition is output), and conversely the rule for $a \,\mathsf{M}\, b$ is the same as R with the promise of a.

The *weak until* operator can be used to express properties such as φ becomes true before ψ or at the same time: $(\neg\psi) \,\mathsf{W}\, \varphi$. Such pattern often occur in the LTL formulæ presented by Dwyer et al. (1998) and are encoded with the U operator[15]. Since W can be translated without promises or acceptance conditions, it makes sense to rewrite heavier patterns such as $a \,\mathsf{U}(b \vee \mathsf{G}\, a)$ using W prior to the traduction. We implemented the following rules in Spot-0.6 to that effect:

[15]The $a \,\mathsf{U}(b \vee \mathsf{G}\, a)$ idiom was used. The LTL formulæ on the web page associated to their project, at `http://patterns.projects.cis.ksu.edu/documentation/patterns/ltl.shtml`, have since been rewritten with the W operator for clarity.

$$
\begin{array}{rcl}
a \,\mathsf{U}(b \vee \mathsf{G}\, a) = a \,\mathsf{W}\, b & & a \,\mathsf{R}(b \wedge \mathsf{F}\, a) = a \,\mathsf{M}\, b \\
a \,\mathsf{W}(b \vee \mathsf{G}\, a) = a \,\mathsf{W}\, b & & a \,\mathsf{M}(b \wedge \mathsf{F}\, a) = a \,\mathsf{M}\, b \\
(a \,\mathsf{U}\, b) \vee \mathsf{G}\, a = a \,\mathsf{W}\, b & & a \,\mathsf{R}\, \mathsf{F}\, a = \mathsf{F}\, a \\
(a \,\mathsf{U}\, b) \vee (a \,\mathsf{W}\, c) = a \,\mathsf{W}\, (b \vee c) & & a \,\mathsf{M}\, \mathsf{F}\, a = \mathsf{F}\, a \\
(a \,\mathsf{W}\, b) \vee (a \,\mathsf{W}\, c) = a \,\mathsf{W}\, (b \vee c) & & (a \,\mathsf{R}\, b) \wedge \mathsf{F}\, a = a \,\mathsf{M}\, b \\
a \,\mathsf{U}\, \mathsf{G}\, a = \mathsf{G}\, a & & (a \,\mathsf{R}\, b) \wedge (a \,\mathsf{M}\, c) = a \,\mathsf{M}(b \wedge c) \\
a \,\mathsf{W}\, \mathsf{G}\, a = \mathsf{G}\, a & & (a \,\mathsf{M}\, b) \wedge (a \,\mathsf{M}\, c) = a \,\mathsf{M}(b \wedge c) \\
(a \,\mathsf{U}\, b) \wedge (c \,\mathsf{W}\, b) = (a \wedge c) \,\mathsf{U}\, b & & (a \,\mathsf{R}\, b) \vee \mathsf{G}\, b = a \,\mathsf{R}\, b \\
(a \,\mathsf{W}\, b) \wedge (c \,\mathsf{W}\, b) = (a \wedge c) \,\mathsf{W}\, b & & (a \,\mathsf{M}\, b) \vee \mathsf{G}\, b = a \,\mathsf{R}\, b \\
(a \,\mathsf{R}\, b) \vee (c \,\mathsf{M}\, b) = (a \vee c) \,\mathsf{R}\, b & & (a \,\mathsf{U}\, b) \wedge \mathsf{F}\, b = a \,\mathsf{U}\, b \\
(a \,\mathsf{M}\, b) \vee (c \,\mathsf{M}\, b) = (a \vee c) \,\mathsf{M}\, b & & (a \,\mathsf{W}\, b) \wedge \mathsf{F}\, b = a \,\mathsf{U}\, b
\end{array}
$$

$$
\begin{array}{rl}
a \,\mathsf{M}(b \,\mathsf{M}\, c) = a \,\mathsf{M}\, c & \text{if } a \Rightarrow b. \\
a \,\mathsf{M}(b \,\mathsf{R}\, c) = a \,\mathsf{M}\, c & \text{if } a \Rightarrow b. \\
a \,\mathsf{R}(b \,\mathsf{R}\, c) = a \,\mathsf{R}\, c & \text{if } a \Rightarrow b. \\
a \,\mathsf{R}(b \,\mathsf{M}\, c) = b \,\mathsf{M}\, c & \text{if } b \Rightarrow a. \\
a \,\mathsf{M}(b \,\mathsf{M}\, c) = b \,\mathsf{M}\, c & \text{if } b \Rightarrow a.
\end{array}
$$

The 11 formulæ for which the translation was improved correspond to formulæ of Dwyer et al. (1998) in which the $\varphi \,\mathsf{U}(\psi \vee \mathsf{G}\, \varphi)$ idiom was used. Rewriting these using W helped the translation since it did not have to deal with promises associated to U.

4.3. Reaching the Minimum

Cichoń et al. (2009) studied several classes of LTL formulæ for which they calculated the size (in states) of the minimal Büchi automaton that could represent the property. They compared the output of spot-0.4 and ltl2ba-1.1, neither of which was able translate all formulæ efficiently. Sometimes they would take too long, sometimes they would produce automata larger than necessary.

Here are four families of formulæ they evaluated on both tools for n ranging from 1 to 20:

$$\alpha_n = \mathsf{F}(\underbrace{p \wedge \mathsf{X}(p \wedge X(p \wedge \ldots))}_{n \text{ occurences of } p}) \wedge \mathsf{F}(\underbrace{q \wedge \mathsf{X}(q \wedge X(q \wedge \ldots))}_{n \text{ occurences of } q})$$

$$\beta_n = \mathsf{F}(p_1 \wedge \mathsf{F}(p_2 \wedge \mathsf{F}(\ldots \mathsf{F}\, p_n))) \wedge \mathsf{F}(q_1 \wedge \mathsf{F}(q_2 \wedge \mathsf{F}(\ldots \mathsf{F}\, q_n)))$$

$$\varphi_n = \mathsf{G}\,\mathsf{F}\, p_1 \wedge \mathsf{G}\,\mathsf{F}\, p_2 \wedge \ldots \wedge \mathsf{G}\,\mathsf{F}\, p_n$$

$$\psi_n = \mathsf{F}\,\mathsf{G}\, p_1 \vee \mathsf{F}\,\mathsf{G}\, p_2 \vee \ldots \vee \mathsf{F}\,\mathsf{G}\, p_n$$

The minimal Büchi automaton for α_n has $(n+1)^2$ states. ltlba-1.1 and spot-0.4 both manage to produce it, but they take a lot of time. In their experiment ltl2ba-1.1 took 14 hours to translate α_{19} (they did not try α_{20}) while spot-0.4 took 6 seconds to translate α_{20}. But 6 seconds are a lot for such a formula: spot-0.7.1 takes less than 0.01 second.

The minimal Büchi automaton for β_n also has $(n+1)^2$ states, but a lot more transitions. Neither ltl2ba-1.1 nor spot-0.4 were able to produce the smallest automata for $n > 1$. For β_{20} the minimal automaton has 441 states; ltl2ba-1.1 produced an automaton of

1164 states in 21 seconds while spot-0.4 produced 1237 states in about 12 minutes.

φ_n did not cause any problem with either tools. The minimal automaton were produced by both tools instantaneously.

The worse formula was ψ_n although both tools manage to build the minimal automaton (with $n+1$ states), they have to deal with an exponential number of transitions. ltl2ba took more that 3 hours to translate ψ_{11} while spot-0.4 took 20 minutes for ψ_{18} and subsequently died on ψ_{19} (out-of-memory?).

We are happy to report that spot-0.7.1 is able to run this entire benchmark (the four families with n ranging from 1 to 20) in less than 15 minutes and always produces Büchi automata with the minimal number of states. The necessary fixes are those described in section 4.2, but running this benchmark on spot-0.6 revealed that the "*pulling of acceptance conditions*" could use a cache to avoid a quadratic behavior[16]. This cache was added in spot-0.7.1.

4.4. WDBA Minimization

It is well known that not all Büchi automata are determinizable (Vardi 1996, prop. 8). There is a subclass of properties that can be represented by Weak Deterministic Büchi Automata (WDBA), and for which there exists an algorithm to compute the minimal WDBA recognizing the property in a way that is comparable to the minimization of deterministic finite automata (Löding 2001). This class corresponds to the "*obligations*" in the temporal hierarchy of Manna and Pnueli (1990) and includes a large number of LTL formulæ used for model checking (starting with all safety properties). For instance 40 formulæ out of the 55 formulæ from Dwyer et al. (1998) are obligations.

Dax et al. (2007) did a comparison of the size produced by different translators (not Spot, which they did not know) with the size of the minimal WDBA. This revealed that although it was deterministic, the minimal WDBA usually had number states smaller or equal to that of the automata produced by the translators.

This WDBA minimization has since been integrated into spot-0.7.1. Table 3 completes the benchmark of Dax et al. (2007) using the sizes of the degeneralized automata generated by spot-0.7.1. The first column is the number of the formula, so you can compare with the figure for other tools displayed at `http://www.daxc.de/eth/atva07/index.html`. The second and third numbers give the number of states and transitions[17] of the automaton produced by Spot (with formula simplifications and SCC simplifications turned on), while the fourth and fifth numbers show the number of states and transitions with an additional WDBA minimization step.

[16] This is because the degeneralization is performed on-the-fly in Spot: the degeneralized automaton is constructed from the original TGBA as needed by subsequent algorithms and never stored.

We can observe that some minimized automata have more transitions: this is because their structure changed when they were determinized. Even though they have the same number of states as the non-minimized automaton, the states do not accept the same language. There is even one case (formula 36) where the minimized automaton got one more state.

In only two cases (formulae 31 and 35) the minimization actually removed states in addition to making the automata deterministic.

5. CONCLUSION

We have presented the main improvements realized in the LTL translation module of Spot between versions 0.4 and 0.7.1, both in the size of the produced automata and in the time it takes to generate them. We also gave insight into its implementation and explained why using BDDs during the translation actually allows many reduction.

It has been argued (Cichoń et al. 2009; Tsay et al. 2011) that rather than optimizing an algorithm to try to produce the best automata always, it would be useful to create a database of optimal automata for commonly used formulæ. However different uses may call for different definition of *optimal automaton*. In the context of model-checking, one usually wants to reduce the size of product of the property with the system, and translating the property into a small automaton that is the most deterministic possible usually helps (Sebastiani and Tonetta 2003). In the context of synthesis of reactive systems Ehlers and Finkbeiner (2010) prefer to minimize the number of states at the expense of determinism.

Also different kinds of automata can be used for verification: model checking with TGBA is usually better than model checking with Büchi automata when the formula incur a lot of acceptance conditions (Couvreur et al. 2005). Using testing automata also appears promising (Geldenhuys and Hansen 2006; Ben Salem et al. 2011). A database should therefore not be limited to Büchi automata.

While we agree such a database could indeed be useful, we still believe that it is important to have a translation that is efficient and versatile enough to be tuned to the needs of a particular situation.

[17] See footnote 14.

	before		after		
#	st.	tr.	st.	tr.	formula
1	2	4	2	4	$\neg(\mathsf{G}\,\bar{p})$
2	3	10	3	10	$\neg(\mathsf{F}\,r \rightarrow (\bar{p} \mathsf{U} r))$
3	3	13	3	12	$\neg(\mathsf{G}(q \rightarrow \mathsf{G}\,\bar{p}))$
4	4	30	4	32	$\neg(\mathsf{G}((q \wedge \bar{r} \wedge \mathsf{F}\,r) \rightarrow (\bar{p} \mathsf{U} r)))$
5	3	21	3	24	$\neg(\mathsf{G}(q \wedge \bar{r} \rightarrow ((\bar{p} \mathsf{U} r) \vee \mathsf{G}\,\bar{p})))$
6	1	1	1	1	$\neg(\mathsf{F}\,p)$
7	2	7	2	7	$\neg((\bar{r} \mathsf{U}(p \wedge \bar{r})) \vee (\mathsf{G}\,\bar{r}))$
8	2	5	2	5	$\neg(\mathsf{G}\,\bar{q} \vee \mathsf{F}(q \wedge \mathsf{F}\,p))$
9	3	23	3	24	$\neg(\mathsf{G}(q \wedge \bar{r} \rightarrow ((\bar{r} \mathsf{U}(p \wedge \bar{r})) \vee \mathsf{G}\,\bar{r})))$
10	6	12	6	12	$\neg((\bar{p} \mathsf{U}((p \mathsf{U}((\bar{p} \mathsf{U}((p \mathsf{U} \mathsf{G}\,\bar{p}) \vee \mathsf{G}\,p)) \vee \mathsf{G}\,\bar{p})) \vee \mathsf{G}\,p)) \vee \mathsf{G}\,\bar{p})$
11	7	18	7	18	$\neg(\mathsf{F}\,r \rightarrow ((\bar{p} \wedge \bar{r}) \mathsf{U}(r \vee ((p \wedge \bar{r}) \mathsf{U}(r \vee ((\bar{p} \wedge \bar{r}) \mathsf{U}(r \vee ((p \wedge \bar{r}) \mathsf{U}(r \vee (\bar{p} \mathsf{U} r))))))))))$
12	7	28	7	28	$\neg(\mathsf{F}\,q \rightarrow (\bar{q} \mathsf{U}(q \wedge ((\bar{p} \mathsf{U}((p \mathsf{U}((\bar{p} \mathsf{U}((p \mathsf{U} \mathsf{G}\,\bar{p}) \vee \mathsf{G}\,p)) \vee \mathsf{G}\,\bar{p})) \vee \mathsf{G}\,p)) \vee \mathsf{G}\,\bar{p}))))$
13	8	46	8	64	$\neg(\mathsf{G}((q \wedge \mathsf{F}\,r) \rightarrow ((\bar{p} \wedge \bar{r}) \mathsf{U}(r \vee ((p \wedge \bar{r}) \mathsf{U}(r \vee ((\bar{p} \wedge \bar{r}) \mathsf{U}(r \vee ((p \wedge \bar{r}) \mathsf{U}(r \vee (\bar{p} \mathsf{U} r)))))))))))$
14	7	38	7	56	$\neg(\mathsf{G}(q \rightarrow ((\bar{p} \wedge \bar{r}) \mathsf{U}(r \vee ((p \wedge \bar{r}) \mathsf{U}(r \vee ((\bar{p} \wedge \bar{r}) \mathsf{U}(r \vee ((p \wedge \bar{r}) \mathsf{U}(r \vee ((\bar{p} \mathsf{U} r) \vee \mathsf{G}\,\bar{p}) \vee \mathsf{G}\,p))))))))))$
15	2	4	2	4	$\neg(\mathsf{G}\,p)$
16	3	10	3	10	$\neg(\mathsf{F}\,r \rightarrow (p \mathsf{U} r))$
17	3	13	3	12	$\neg(\mathsf{G}(q \rightarrow \mathsf{G}\,p))$
18	4	15	4	16	$\neg(\mathsf{G}((p \wedge \bar{r} \wedge \mathsf{F}\,r) \rightarrow (p \mathsf{U} r)))$
19	3	21	3	24	$\neg(\mathsf{G}(q \wedge \bar{r} \rightarrow ((p \mathsf{U} r) \vee \mathsf{G}\,p)))$
20	4	12	4	12	$\neg((\bar{p} \mathsf{U} s) \vee \mathsf{G}\,p)$
21	3	18	3	18	$\neg(\mathsf{F}\,r \rightarrow (\bar{p} \mathsf{U}(s \vee r)))$
22	4	54	4	64	$\neg(\mathsf{G}((q \wedge \bar{r} \wedge \mathsf{F}\,r) \rightarrow (\bar{p} \mathsf{U}(s \vee r))))$
23	3	37	3	48	$\neg(\mathsf{G}(q \wedge \bar{r} \rightarrow ((\bar{p} \mathsf{U}(s \vee r)) \vee \mathsf{G}\,\bar{p})))$
24	3	19	3	20	$\neg(\mathsf{F}\,r \rightarrow (p \rightarrow (\bar{r} \mathsf{U}(s \wedge \bar{r}))) \mathsf{U} r)$
25	4	59	4	64	$\neg(\mathsf{G}((q \wedge \bar{r} \wedge \mathsf{F}\,r) \rightarrow (p \rightarrow (\bar{r} \mathsf{U}(s \wedge \bar{r}))) \mathsf{U} r))$
26	3	20	3	20	$\neg(\mathsf{F}\,p \rightarrow (\bar{p} \mathsf{U}(s \wedge \bar{p} \wedge \mathsf{X}(\bar{p} \mathsf{U} t))))$
27	4	44	4	44	$\neg(\mathsf{F}\,r \rightarrow (\bar{p} \mathsf{U}(r \vee (s \wedge \bar{p} \wedge \mathsf{X}(\bar{p} \mathsf{U} t)))))$
28	4	48	4	48	$\neg((\mathsf{G}\,\bar{q}) \vee (\bar{q} \mathsf{U}(q \wedge \mathsf{F}\,p \rightarrow (\bar{p} \mathsf{U}(s \wedge \bar{p} \wedge \mathsf{X}(\bar{p} \mathsf{U} t))))))$
29	5	128	5	160	$\neg(\mathsf{G}((q \wedge \mathsf{F}\,r) \rightarrow (\bar{p} \mathsf{U}(r \vee (s \wedge \bar{p} \wedge \mathsf{X}(\bar{p} \mathsf{U} t))))))$
30	4	92	4	128	$\neg(\mathsf{G}(q \rightarrow (\mathsf{F}\,p \rightarrow (\bar{p} \mathsf{U}(r \vee (s \wedge \bar{p} \wedge \mathsf{X}(\bar{p} \mathsf{U} t)))))))$
31	4	34	3	20	$\neg((\mathsf{F}(s \wedge \mathsf{X}\,\mathsf{F}\,t)) \rightarrow (\bar{s} \mathsf{U} p))$
32	4	46	4	44	$\neg(\mathsf{F}\,r \rightarrow ((\neg(s \wedge \bar{r} \wedge \mathsf{X}(\bar{r} \mathsf{U}(t \wedge \bar{r})))) \mathsf{U}(r \vee p)))$
33	5	82	4	52	$\neg((\mathsf{G}\,\bar{q}) \vee (\bar{q} \mathsf{U}(q \wedge ((\mathsf{F}(s \wedge \mathsf{X}\,\mathsf{F}\,t)) \rightarrow (\bar{s} \mathsf{U} p)))))$
34	5	130	5	160	$\neg(\mathsf{G}((q \wedge \mathsf{F}\,r) \rightarrow ((\neg(s \wedge \bar{r} \wedge \mathsf{X}(\bar{r} \mathsf{U}(t \wedge \bar{r})))) \mathsf{U}(r \vee p))))$
35	10	254	4	128	$\neg(\mathsf{G}(q \rightarrow (\neg(s \wedge \bar{r} \wedge \mathsf{X}(\bar{r} \mathsf{U}(t \wedge \bar{r})))) \mathsf{U}(r \vee p) \vee \mathsf{G}(\neg(s \wedge \mathsf{X}\,\mathsf{F}\,t)))))$
36	4	36	5	50	$\neg(\mathsf{F}\,r \rightarrow (s \wedge \mathsf{X}(\bar{r} \mathsf{U} t) \rightarrow \mathsf{X}(\bar{r} \mathsf{U}(t \wedge \mathsf{F}\,p))) \mathsf{U} r)$
37	4	52	4	52	$\neg(\mathsf{F}\,r \rightarrow (p \rightarrow (\bar{r} \mathsf{U}(s \wedge \bar{r} \wedge \mathsf{X}(\bar{r} \mathsf{U} t)))) \mathsf{U} r)$
38	5	148	5	160	$\neg(\mathsf{G}((q \wedge \mathsf{F}\,r) \rightarrow (p \rightarrow (\bar{r} \mathsf{U}(s \wedge \bar{r} \wedge \mathsf{X}(\bar{r} \mathsf{U} t)))) \mathsf{U} r))$
39	4	104	4	104	$\neg(\mathsf{F}\,r \rightarrow (p \rightarrow (\bar{r} \mathsf{U}(s \wedge \bar{r} \wedge \bar{z} \wedge \mathsf{X}((\bar{r} \wedge \bar{z}) \mathsf{U} t)))) \mathsf{U} r)$
40	5	296	5	320	$\neg(\mathsf{G}((q \wedge \mathsf{F}\,r) \rightarrow (p \rightarrow (\bar{r} \mathsf{U}(s \wedge \bar{r} \wedge \bar{z} \wedge \mathsf{X}((\bar{r} \wedge \bar{z}) \mathsf{U} t)))) \mathsf{U} r))$

Table 3: *Size of (degeneralized) Büchi automata produced by spot-0.7.1 before and after WDBA minimization.*

For the interested reader, the *Büchi Store* project (Tsay et al. 2011) is a database of formulæ associated to representative Büchi automata, where users can submit better automata. In February 2011, we downloaded their list of formulæ and automata for benchmarking purpose. Out of the 322 formulæ involving only future-time LTL operators Spot was able to produce smaller automata in 52 cases, and bigger automata in 21 cases (in this comparison, the number of transitions was used to compare automata with equal number of states). We have yet to submit our automata for these 52 cases.

ACKNOWLEDGMENTS

The author would like to thank Kristin Rozier (Robust Software Engineering team at NASA), Rüdiger Ehlers (Saarland University), and Christian Dax (formerly at ETH Zürick) for fruitful discussions and for sharing their tools to (respectively) generate C_n formulæ, compare translators, and minimize WDBAs. Felix Abecassis (Epita student) wrote a large part of the WDBA minimization in Spot. Denis Poitrenaud (LIP6) reported several mistakes in draft versions of this paper.

REFERENCES

Ben Salem, A. E., Duret-Lutz, A. and Kordon, F. (2011), Generalized Büchi automata versus testing automata for model checking, In *Proc. of SUMO'11*, Vol. 626 of *Workshop Proceedings*, CEUR.

Bloem, R. (2001), Search Techniques and Automata for Symbolic Model Checking, PhD thesis, University of Colorado.

Bryant, R. E. (1986), 'Graph-based algorithms for boolean function manipulation', *IEEE Transactions on Computers* **35**(8), 677–691.

Cichoń, J., Czubak, A. and Jasiński, A. (2009), Minimal Büchi automata for certain classes of LTL formulas, In *Proc. of DEPCOS'09*, IEEE Computer Society, pp. 17–24.

Clarke, E. M., Grumberg, O. and Peled, D. A. (2000), *Model Checking*, The MIT Press.

Couvreur, J.-M. (1999), On-the-fly verification of temporal logic, In *Proc. of FM'99*, Vol. 1708 of *LNCS*, Springer, pp. 253–271.

Couvreur, J.-M., Duret-Lutz, A. and Poitrenaud, D. (2005), On-the-fly emptiness checks for generalized Büchi automata, In *Proc. of SPIN'05*, Vol. 3639 of *LNCS*, Springer, pp. 143–158.

Dax, C., Eisinger, J. and Klaedtke, F. (2007), Mechanizing the powerset construction for restricted classes of ω-automata, In *Proc. of ATVA'07*, Vol. 4762 of *LNCS*, Springer.

Duret-Lutz, A. and Poitrenaud, D. (2004), SPOT: an Extensible Model Checking Library using Transition-based Generalized Büchi Automata, In *Proc. of MASCOTS'04*, IEEE Computer Society Press, pp. 76–83.

Dwyer, M. B., Avrunin, G. S. and Corbett, J. C. (1998), Property specification patterns for finite-state verification, In *Proc. of FMSP'98*, ACM Press, pp. 7–15.

Ehlers, R. and Finkbeiner, B. (2010), On the virtue of patience: minimizing Büchi automata, In *Proc. of SPIN'10*, Vol. 6349 of *LNCS*, Springer, pp. 129–145.

Etessami, K. and Holzmann, G. J. (2000), Optimizing Büchi automata, In *Proc. of Concur'00*, Vol. 1877 of *LNCS*, Springer, pp. 153–167.

Etessami, K., Wilke, T. and Schuller, R. A. (2001), Fair simulation relations, parity games, and state space reduction for Büchi automata, In *Proc. of the 28th international colloquium on Automata, Languages and Programming*, Vol. 2076 of *LNCS*, Springer, pp. 694–707.

Gastin, P. and Oddoux, D. (2001), Fast LTL to Büchi automata translation, In *Proc. of CAV'01*, Vol. 2102 of *LNCS*, Springer, pp. 53–65.

Geldenhuys, J. and Hansen, H. (2006), Larger automata and less work for LTL model checking, In *Proc. of SPIN'06*, Vol. 3925 of *LNCS*, Springer, pp. 53–70.

Giannakopoulou, D. and Lerda, F. (2002), From states to transitions: Improving translation of LTL formulæ to Büchi automata, In *Proc. of FORTE'02*, Vol. 2529 of *LNCS*, Springer, pp. 308–326.

Löding, C. (2001), 'Efficient minimization of deterministic weak ω-automata', *Information Processing Letters* **79**(3), 105–109.

Manna, Z. and Pnueli, A. (1990), A hierarchy of temporal properties, In *Proc. of PODC'90*, ACM, pp. 377–410.

Minato, S. (1992), Fast generation of irredundant sum-of-products forms from binary decision diagrams, In *Proc. of SASIMI'92*, pp. 64–73.

Oddoux, D. (2003), Utilisation des automates alternants pour un model-checking efficace des logiques temporelles linéaires, PhD thesis, Universitée Paris 7, Paris, France.

Rozier, K. Y. and Vardi, M. Y. (2007), LTL satisfiability checking, In *Proc. of SPIN'07*, Vol. 4595 of *LNCS*, Springer, pp. 149–167.

Sebastiani, R. and Tonetta, S. (2003), "more deterministic" vs. "smaller" Büchi automata for efficient LTL model checking, In *Proc. of CHARME'03*, Vol. 2860 of *LNCS*, Springer, pp. 126–140.

Somenzi, F. and Bloem, R. (2000), Efficient Büchi automata for LTL formulæ, In *Proc. of CAV'00*, Vol. 1855 of *LNCS*, Springer, pp. 247–263.

Tauriainen, H. (2006), Automata and Linear Temporal Logic: Translation with Transition-based Acceptance, PhD thesis, Helsinki University of Technology, Espoo, Finland.

Tsay, Y.-K., Tsai, M.-H., Chang, J.-S. and Chang, Y.-W. (2011), Büchi store: An open repository of büchi automata, In *Proc. of TACAS'11*, Vol. 6605 of *LNCS*, Springer, pp. 262–266.

Vardi, M. Y. (1996), An automata-theoretic approach to linear temporal logic, In *Proc. of Banff'94*, Vol. 1043 of *LNCS*, Springer, pp. 238–266.

Vardi, M. Y. (2007), Automata-theoretic model checking revisited, In *Proc. of VMCAI'07*, Vol. 4349 of *LNCS*, Springer. Invited paper.