SAT-based Minimization of Deterministic ω-Automata

Souheib Baarir and Alexandre Duret-Lutz

LRDE, EPITA, Le Kremlin-Bicêtre, France
sbaarir@lrde.epita.fr,adl@lrde.epita.fr

Abstract. We describe a tool that inputs a deterministic ω-automaton with any acceptance condition, and synthesizes an equivalent ω-automaton with another arbitrary acceptance condition and a given number of states, if such an automaton exists. This tool, that relies on a SAT-based encoding of the problem, can be used to provide minimal ω-automata equivalent to given properties, for different acceptance conditions.

1 Introduction

LTL Synthesis and Probabilistic LTL Model Checking (PMC) are two areas where it is useful to express linear-time temporal properties as deterministic ω-automata. Because it is well known that not all Büchi automata can be made deterministic, these applications use other acceptance conditions such a Rabin or Streett. The model checker `PRISM` [12], for instance, contains a reimplementation of `ltl2dstar` [8], a tool that converts non-deterministic Büchi automata (obtained from an LTL formula) into deterministic Rabin or Streett automata, using Safra's construction [14].

In the past few years, there have been a blossoming of tools directly translating LTL formulas into Rabin automata, or generalized variants of Rabin automata [10, 3, 11, 5, 9]. These tools usually give automata smaller than those obtained with `ltl2dstar` via Safra's construction, and it has been shown that using the generalized Rabin condition can speed PMC up by orders of magnitude [5, 9].

The need for interaction between tools producing and consuming ω-automata with various acceptance conditions has led to the introduction of the Hanoi Omega-Automata (HOA) format [4], where the acceptance condition can be specified using an arbitrary Boolean expression of sets that must be visited infinitely often or finitely often. The current implementation of `PRISM` can perform PMC using deterministic automata having any such arbitrary acceptance condition, and to save memory it is preferable to have automata with as few states as possible, even if this means having a more complex acceptance condition.

In this paper, we present a tool that inputs a deterministic automaton with any acceptance condition, and uses a SAT-based technique to synthetize an equivalent automaton with any given accepance condition and number of states if such an automaton exists. As a consequence we also have a way to construct minimal equivalent deterministic automata for any given acceptance condition.

This SAT-based encoding is costly, so it is not suitable for routine simplification of automata; however it is a very useful tool to provide lowerbounds for the size of the deterministic automata that existing LTL translators (or actually, any automaton transformation tool) could produce, so it should help authors of such tools to find cases where there is room for improvement.

The SAT-based encoding we use for this synthesis with any acceptance is an extension of our previous work that was restricted to generalized-Büchi acceptance [2], and that was itself a generalization of the `DBAminimizer` of Ehlers [7] for Büchi acceptance.

2 Definitions and Encoding

2.1 Deterministic Transition-based ω-Automaton

For a set S, S^ω denotes the set of infinite sequences over S. Given such an infinite sequence $\sigma \in S^\omega$, $\mathrm{Inf}(\sigma)$ denotes the subset of elements that occur infinitely often in σ. We use $\mathbb{B} = \{\bot, \top\}$ to denote the set of Boolean constants, and use $[m]$ as a shorthand for $\{1, 2, \ldots, m\}$.

Definition 1 (DTωA). *A (complete) Deterministic Transition-based ω-Automaton (DTωA) is a tuple $\mathcal{A} = \langle Q, \Sigma, \iota, \delta, (F_1, F_2, \ldots, F_m), \mathscr{F}\rangle$ where*

- *Q is a set of states, $\iota \in Q$ is the initial state,*
- *Σ is an alphabet,*
- *$\delta \subseteq Q \times \Sigma \times Q$ is a transition relation that is deterministic and complete, i.e., such that $\forall (s, \ell) \in Q \times \Sigma, |\{d \in Q \mid (s, \ell, d) \in \delta\}| = 1$. By abuse of notation, we shall also write $\delta(s)$ to denote the set $\{(\ell, d) \in \Sigma \times Q \mid (s, \ell, d) \in \delta\}$.*
- *$(F_1, F_2, \ldots, F_m)$ is a tuple of m* acceptance sets *of transitions $F_i \subseteq \delta$. For convenience, we denote $\widetilde{F}(t) = \{i \in [m] \mid t \in F_i\}$ the set of indices of acceptance sets that contain t.*
- *$\mathscr{F} : 2^{[m]} \to \mathbb{B}$ is a Boolean function that tells which combination of acceptance sets should be visited infinitely often along a run for this run to be accepting.*

A run *of $\mathcal{A}$ is an infinite sequence of connected transitions $\rho = (q_1, \ell_1, q_2)(q_2, \ell_2, q_3)(q_3, \ell_3, q_4)\ldots \in \delta^\omega$ such that $q_1 = \iota$. This run* recognizes *the infinite word $\ell_1\ell_2\ell_3\ldots$ and is* accepting *iff $\mathscr{F}(\bigcup_{t\in \mathrm{Inf}(\rho)} \widetilde{F}(t)) = \top$. The* language *of $\mathcal{A}$ is the set $\mathscr{L}(\mathcal{A})$ of all infinite words recognized by accepting runs of $\mathcal{A}$.*

For brevity, in the rest of this article we simply write *automaton* instead of *complete and deterministic ω-automaton*.

In the HOA format [4], the acceptance function $\mathscr{F}$ is represented by a Boolean expression over primitives of the form $\mathsf{Inf}(i)$ or $\mathsf{Fin}(i)$ meaning respectively that the set F_i has to be visited infinitely often or finitely often. For instance $\mathsf{Fin}(0) \vee \mathsf{Inf}(1)$ is an expression for Streett acceptance with one pair (a run is accepting if it either visits F_0 finitely often, or F_1 infinitely often); in our definition of DTωA, the corresponding $\mathscr{F}$ function would be such that $\mathscr{F}(\{1\}) = \mathscr{F}(\{0, 1\}) = \mathscr{F}(\emptyset) = \top$, and $\mathscr{F}(\{0\}) = \bot$.

2.2 Synthesis of Equivalent DTωA

Given an automaton $R = \langle Q_R, \Sigma, \iota_R, \delta_R, (F_1, F_2, \ldots, F_{m'}), \mathscr{F}\rangle$, two integers n and m, and an acceptance function $\mathscr{G} : 2^{[m]} \to \mathbb{B}$, we would like to construct (if it exists) an automaton $C = \langle Q_C, \Sigma, \iota_C, \delta_C, (G_1, G_2, \ldots, G_m), \mathscr{G}\rangle$ with $|Q_C| = n$ states, and such that $\mathscr{L}(R) = \mathscr{L}(C)$. We call R the *reference* automaton, and C, the *candidate* automaton.

Since C and R are complete and deterministic, any word of Σ^ω has a unique run in R and C, and testing $\mathscr{L}(R) = \mathscr{L}(C)$ can be done by ensuring that each word is accepted by R iff it is accepted by C. In practice, this is checked by ensuring that any cycle of the synchronous product $C \otimes R$ corresponds to cycles that are either accepting in both C and R, or rejecting in both. To ensure that property, the SAT-based encoding uses variables to encode the history of acceptance sets visited between two states of the product $C \otimes R$.

SAT Variables We encode C with two sets of variables:

- The "triplet" variables $\{\langle q_1, \ell, q_2\rangle \mid (q_1, q_2) \in Q_C^2,\ \ell \in \Sigma\}$ encode the existence of transitions $(q_1, \ell, q_2) \in \delta_C$ in the candidate automaton.
- The "quadruplet" variables $\{\langle q_1, \ell, i, q_2\rangle \mid (q_1, q_2) \in Q_C^2,\ \ell \in \Sigma, i \in [m]\}$ encode the membership of these transitions to each acceptance set G_i of C.

For the product $C \otimes R$, we encode the reachable states, and parts of paths that might eventually be completed to become cycles. We use $SCC_R \subseteq 2^{Q_R}$ to denote the set of non-trivial strongly connected components of R.

- A variable in $\{\langle q, q', q, q', \emptyset, \emptyset\rangle \mid q \in Q_C, q' \in Q_R\}$ encodes the existence of a reachable state (q, q') in $C \otimes R$. The reason we use a sextuplet to encode such a pair is that each (q, q') will serve as a starting point for possible paths.
- A variable in $\{\langle q_1, q_1', q_2, q_2', I, I'\rangle \mid (q_1, q_2) \in Q_C^2,\ S \in SCC_R,\ (q_1', q_2') \in S^2,\ I \subseteq [m],\ I' \subseteq [m']\}$ denotes that there is a path between (q_1, q_1') and (q_2, q_2') in the product, such that its projection on C visits the acceptance sets G_i for all $i \in I$, and its projection on R visits the acceptance sets F_i for all $i \in I'$. This set of variables is used to implement the cycle equivalence check, so the only q_1' and q_2' that need to be considered should belong to the same non-trivial SCC of R.

SAT Contraints With the above variables, C can be obtained as a solution of the following SAT problem. First, C should be complete (i.e., δ_C is total):

$$\bigwedge_{q_1 \in Q_C,\, \ell \in \Sigma} \bigvee_{q_2 \in Q_C} \langle q_1, \ell, q_2\rangle \tag{1}$$

Then, the initial state of the product must exist. Furthermore, if (q_1, q_1') is a state of the product, $(q_1', \ell, q_2') \in \delta_R$ is a transition in the reference automaton, and $(q_1, \ell, q_2) \in \delta_C$ is a transition in the candidate automaton, then (q_2, q_2') is a state of the product too:

$$\wedge\, \langle \iota_C, \iota_R, \iota_C, \iota_R, \emptyset, \emptyset\rangle \wedge \bigwedge_{\substack{(q_1,q_2)\in Q_C^2,\, q_1' \in Q_R,\\ (\ell, q_2') \in \delta_R(q_1')}} \langle q_1, q_1', q_1, q_1', \emptyset, \emptyset\rangle \wedge \langle q_1, \ell, q_2\rangle \rightarrow \langle q_2, q_2', q_2, q_2', \emptyset, \emptyset\rangle \tag{2}$$

Any transition of the product augments an existing path, updating the sets I and I' of indices of acceptance sets visited in each automaton. Unfortunately, we

have to consider all possible subsets $J \subseteq [m]$ of acceptances sets to which the candidate transition (q_2, ℓ, q_3) could belong, and emit a different rule for each J.

$$\wedge \bigwedge_{\substack{(q_1,q_2,q_3)\in Q_C^3,\\ I\subseteq[m],\, J\subseteq[m],\\ S\in SCC_R,\,(q_1',q_2')\in S^2,\\ I'\subseteq[m'],\,(\ell,q_3')\in\delta_R(q_2')}} \left(\begin{array}{l} \langle q_1, q_1', q_2, q_2', I, I'\rangle \\ \wedge \langle q_2, \ell, q_3\rangle \\ \wedge \bigwedge_{i\in J} \langle q_2, \ell, i, q_3\rangle \\ \wedge \bigwedge_{i\notin J} \neg\langle q_2, \ell, i, q_3\rangle \end{array} \right) \rightarrow \begin{array}{l} \langle q_1, q_1', q_3, q_3', I\cup J, \\ \quad I' \cup \widetilde{F}((q_2', \ell, q_3'))\rangle \end{array} \tag{3}$$

If a path of the product is followed by a transition $(q_2', \ell, q_3') \in \delta_R$ and a transition $(q_2, \ell, q_3) \in \delta_C$ that both close a cycle ($q_3 = q_1 \wedge q_3' = q_1'$), then the cycle formed in the candidate automaton by (q_2, ℓ, q_1) should have the same acceptance (i.e., rejecting or accepting) as the cycle of the reference automaton. In other words, the transition (q_2, ℓ, q_1) belongs to a subset $J \subseteq [m]$ of acceptance sets only if this J satisfies $\mathscr{G}(I \cup J) = \mathscr{F}(I' \cup \widetilde{F}((q_2', \ell, q_1')))$.

$$\wedge \bigwedge_{\substack{(q_1,q_2)\in Q_C^2,\, I\subseteq[m],\\ S\in SCC_R,\,(q_1',q_2')\in S^2,\, I'\subseteq[m'],\\ (\ell,q_3')\in\delta_R(q_2'),\, q_3'=q_1'}} \langle q_1, q_1', q_2, q_2', I, I'\rangle \wedge \langle q_2, \ell, q_1\rangle \rightarrow \bigvee_{\substack{J\subseteq[m]\\ \mathscr{G}(I\cup J)=\mathscr{F}(I'\cup\widetilde{F}((q_2',\ell,q_1')))}} \Big(\bigwedge_{i\in J} \langle q_2, \ell, i, q_1\rangle \wedge \bigwedge_{i\notin J} \neg\langle q_2, \ell, i, q_1\rangle \Big) \tag{4}$$

Optimizations. A first optimization is to use the same symmetry breaking clauses as suggested by Ehlers [7], to restrict the search space of the SAT solver. Nonetheless, the above encoding requires $\mathrm{O}(|Q_R|^2 \times |Q_C|^2 \times 2^{m+m'})$ variables and $\mathrm{O}(|Q_R|^2 \times |Q_C|^3 \times 2^{2m+m'} \times |\Sigma|)$ clauses. It is therefore very sensitive to the number of acceptance sets used in the reference and candidate automata. To mitigate this, we implement some additional optimizations:

1. For SCCs that are known to be weak (i.e., all cycles are accepting, or all cycles are rejecting) it is not necessary to remember the history I' of the acceptance sets seen by paths. The sets of variables $\{\langle q_1, q_1', q_2, q_2', I, I'\rangle, \ldots\}$ when q_1' and q_2' belong to a weak SCC can therefore be restricted to only cases where $I' = \emptyset$.
2. In case an SCC S is not weak, it is possible that it does not intersect all the sets F_1, F_2, …, F_m. Then the variables $\{\langle q_1, q_1', q_2, q_2', I, I'\rangle, \ldots\}$ can have their history I' restricted to the subset of $[m']$ that actually intersects S.
3. Simplifying histories. Consider a Rabin acceptance condition like $\mathsf{Fin}(0) \wedge \mathsf{Inf}(1)$, where the set F_0 has to be visited finitely often and F_1 has to be visited infinitely often. The histories $I \subseteq [2]$ or $I' \subseteq [2]$ involved in the variables $\{\langle q_1, q_1', q_2, q_2', I, I'\rangle, \ldots\}$ could take any value in $\{\}$, $\{0\}$, $\{1\}$, or $\{0, 1\}$ depending on which sets have been seen along this path. However these variables are only used to detect cycles, and a cycle that contains 0 in its history cannot be prolonged into an accepting cycle: the history $\{0, 1\}$ can therefore be simplified into $\{0\}$, which is enough to ensure that the cycle will be rejecting. Doing this reduces the number of variables and clauses needed.
4. Equation (4) is not directly expressed as a disjunction. To encode it more efficiently, we use BDDs to express the right-hand side of the implication as an irredundant product of sums: depending on whether $\mathscr{F}(I' \cup \widetilde{F}((q_2', \ell, q_1'))$

is accepting, we encode the formula $\mathscr{G}$ or its negation as a BDD, assign to true the BDD variables corresponding to the sets listed in I and obtain the resulting product of sums by dualizing the Minato-Morreale algorithm [13].

State-based acceptance. This encoding can be tweaked to synthetize automata with state-based acceptance by reducing the quadruplets $\langle q_1, \ell, i, q_2\rangle$ to pairs $\langle q_1, i\rangle$ in all the above rules.

3 Implementation and Experiments

3.1 Tool Support

The above encoding is implemented in Spot 1.99.4[1], and can be used via the command-line tool `autfilt`. Tools that produce deterministic ω-automata, such as `ltl2dstar` [8], `ltl3dra` [3], and `Rabinizer3` [9], have all been recently updated to support the Hanoi Omega-Automata format [4], that `autfilt` can input.

The following example translates $\mathsf{G}p_0 \vee \mathsf{FG}p_1$ using `ltl2dstar`. The formula is first passed to `ltlfilt` [6] for conversion into `ltl2dstar`'s input syntax. `ltl2dstar` outputs its result in `dra.hoa`: it is a 5-state Rabin automaton with two pairs of acceptance sets.[2]

```
% ltlfilt -f 'Gp0 | FGp1' -l | ltl2dstar --output-format=hoa - dra.hoa
% egrep 'States:|acc-name:|Acceptance:' dra.hoa
States: 5
acc-name: Rabin 2
Acceptance: 4 (Fin(0)&Inf(1))|(Fin(2)&Inf(3))
```

Now we can minimize this automaton using our SAT-based approach. We pass the `dra.hoa` to `autfilt --sat-minimize`, with additional options to require a complete automaton (`-C`) with state-based acceptance (`-S`), in the HOA format (`-H`). The result has only 3 states.

```
% autfilt -S -C --sat-minimize -H dra.hoa > dra-min.hoa
% egrep 'States:|acc-name:|Acceptance:' dra-min.hoa
States: 3
acc-name: Rabin 2
Acceptance: 4 (Fin(0) & Inf(1)) | (Fin(2) & Inf(3))
```

The `--sat-minimize` option can take additional parameters, for instance to force a particular acceptance condition on the output (the default is the same as for the input). As an example, the following command forces the production of a minimal equivalent automaton with co-Büchi acceptance, which is enough for this formula (and means only one Rabin pair was really necessary).

```
% autfilt -S -C --sat-minimize='acc="co-Buchi"' -H dra.hoa > dra-min1.hoa
% egrep 'States:|acc-name:|Acceptance:' dra-min1.hoa
States: 3
acc-name: co-Buchi
Acceptance: 1 Fin(0)
```

[1] `https://spot.lrde.epita.fr/`

[2] In the HOA format [4] the `Acceptance:` line encodes the $\mathscr{F}$ function of Def. 1, while the `acc-name:` just supplies a human-readable name when one is known.

The `colored` option requests that all transitions (or states) belong to exactly one acceptance set. This is useful for instance when requesting parity acceptance:

```
% autfilt -S -C --sat-minimize='acc="parity␣max␣even␣2",colored' -H \
  dra.hoa >dpa.hoa
% egrep 'States:|acc-name:|Acceptance:' dpa.hoa
States: 3
acc-name: parity max even 2
Acceptance: 2 Fin(1) & Inf(0)
```

One section of the web page `https://spot.lrde.epita.fr/satmin.html` details the usage of `autfilt` with more examples.

3.2 Minimization

To evaluate the usefulness and effectiveness of our tool, we built a benchmark of LTL formulas to convert into deterministic ω-automata by the three translators `ltl2dstar 0.5.3`, `ltl3dra 0.2.2`, and `Rabinizer 3.1`.

Table 1, shows the number states of deterministic Rabin automata produced by the translators, as well as the size of the minimal Rabin automata that `autfilt --sat-minimize` could produce using a single acceptance pair. The table distinguishes the use of state-based acceptance or transition-based acceptance. All automata are complete. Because the SAT encoding is exponential in the number of acceptance sets, but polynomial in the size of the automaton, the input automaton supplied to `autfilt --sat-minimize` was chosen among the automata output by the translators as the one with the fewest number of acceptance sets, and in case of equality the fewest number of states. For instance, for $\neg(\mathsf{GF}p_1 \wedge \mathsf{GF}p_0 \wedge \mathsf{GF}p_2)$ in Table 1, the minimal transition-based automaton of size "3 (2)" was obtained starting from the minimal state-based Rabin automaton of size "4 (2)", not starting from the "1 (6)" automaton produced by `Rabinizer`, as it involves more acceptance sets.

Although this table only shows minimal automata with a single pair, our technique can deal with more pairs (and different acceptance conditions) as well. For instance the formula $(\mathsf{FG}p_0 \vee \mathsf{GF}p_1) \wedge (\mathsf{FG}p_2 \vee \mathsf{GF}p_3)$ is translated by `ltl2dstar` into a DRA with 4385 states (and 7 acceptance pairs), by `ltl3dra` into a DRA with 18 states (and 4 pairs) and by `Rabinizer` into a transition-based DRA with 13 states (and 4 pairs). Using `autfilt --sat-minimize` we could reduce it to a transition-based DRA with 2 states and only 3 acceptance pairs, and show that there is no transition-based DRA with 2 states and less acceptance pairs (the problem becomes unsatisfiable).

Finally Table 2 shows minimization examples that use the transition-based generalized Rabin acceptance introduced by `ltl3dra` and `Rabinizer`. Before minimizing the automaton we simplified the acceptance by removing all unused sets, yielding the simpler acceptance conditions displayed in the table.

Complete results and instructions to reproduce this benchmark can be found at `https://www.lrde.epita.fr/~adl/lpar15/`. In particular, the CSV files include run time information for the SAT solver we used (Glucose 4.0 [1]), and experiments with 2 and 3-pair DRA.

Table 1: Sizes of Rabin automata produced by `ltl2dstar` (L2), `ltl3dra` (L3), `Rabinizer` (R3), or our SAT-based minimization procedure **configured to produce deterministic Rabin automata with a single acceptance pair** (min), with either state or transition-based acceptance. The notation "x (y)" denotes an automaton with x states and y acceptances sets. (In Rabin automata, acceptance sets are used as pairs, so y is always even.) Timeouts after 1h are denoted with "t.o". "imp." (for "impossible") indicates that no Rabin automaton with a single pair where found. Finally "n.a." indicates that the formula falls out of the supported LTL fragment of `ltl3dra`.

	state-based acceptance				tr.-based acc.	
	L2	L3	R3	min	R3	min
$\neg((\mathsf{FG}p_0 \vee \mathsf{GF}p_1) \wedge (\mathsf{FG}p_2 \vee \mathsf{GF}p_3))$	9 (4)	9 (4)	9 (8)	t.o.	4 (4)	imp.
$\neg((\mathsf{GF}p_1 \wedge \mathsf{GF}p_0) \vee (\mathsf{GF}p_3 \wedge \mathsf{GF}p_2))$	270 (10)	10 (8)	11 (16)	imp.	10 (8)	imp.
$\neg\mathsf{F}(\mathsf{G}(p_2 \vee (\overline{p_1} \wedge \overline{p_2})) \vee (\mathsf{X}p_1\,\mathsf{U}(p_0 \wedge \mathsf{X}p_1)))$	9 (2)	n.a.	36 (12)	**5** (2)	34 (6)	**3** (2)
$\neg\mathsf{FG}((p_0 \wedge \mathsf{GF}p_1 \wedge \mathsf{XX}p_1)\,\mathsf{U}\,\mathsf{G}(\mathsf{XX}\overline{p_2} \vee \mathsf{XX}(p_0 \wedge p_1)))$	4 (2)	n.a.	16 (4)	**2** (2)	16 (2)	**1** (2)
$\neg(\mathsf{F}p_0 \wedge (p_1 \vee \mathsf{G}p_2) \wedge (\overline{p_1} \vee \mathsf{F}\overline{p_2}))$	9 (6)	8 (6)	15 (16)	8 (**2**)	13 (8)	8 (**2**)
$\neg(\mathsf{F}p_0 \wedge \mathsf{GF}\overline{p_0})$	4 (4)	3 (4)	4 (8)	3 (**2**)	2 (4)	2 (**2**)
$\neg(\mathsf{F}p_0 \wedge \mathsf{GF}p_1)$	5 (4)	4 (4)	4 (8)	**3** (**2**)	2 (4)	2 (**2**)
$\neg(\mathsf{GF}p_1 \wedge \mathsf{GF}p_0 \wedge \mathsf{GF}p_2)$	8 (6)	8 (6)	8 (12)	**4** (**2**)	1 (6)	3 (**2**)
$\neg(\mathsf{GF}p_1 \wedge \mathsf{GF}p_0)$	4 (4)	4 (4)	4 (8)	**3** (**2**)	1 (4)	2 (**2**)
$\neg\mathsf{GF}p_0$	2 (2)	2 (2)	2 (4)	2 (2)	1 (2)	1 (**2**)
$\neg(\mathsf{X}p_0\,\mathsf{U}\,\mathsf{G}p_1)$	7 (2)	n.a.	6 (8)	**5** (2)	6 (4)	**5** (2)
$(\mathsf{FG}p_0 \vee \mathsf{GF}p_1) \wedge (\mathsf{FG}p_2 \vee \mathsf{GF}p_3)$	4385 (14)	18 (8)	19 (16)	imp.	13 (8)	imp.
$\mathsf{F}(\mathsf{G}(p_2 \vee (\overline{p_1} \wedge \overline{p_2})) \vee (\mathsf{X}p_1\,\mathsf{U}(p_0 \wedge \mathsf{X}p_1)))$	7 (4)	6 (4)	8 (8)	**5** (**2**)	6 (4)	**3** (**2**)
$\mathsf{FG}((p_0 \wedge \mathsf{GF}p_1 \wedge \mathsf{XX}p_1)\,\mathsf{U}\,\mathsf{G}(\mathsf{XX}\overline{p_2} \vee \mathsf{XX}(p_0 \wedge p_1)))$	3 (2)	n.a.	7 (4)	**2** (2)	6 (2)	**1** (2)
$\mathsf{F}p_0 \wedge (p_1 \vee \mathsf{G}p_2) \wedge (\overline{p_1} \vee \mathsf{F}\overline{p_2})$	8 (2)	8 (4)	9 (8)	8 (2)	9 (4)	8 (2)
$\mathsf{F}p_0 \wedge \mathsf{GF}\overline{p_0}$	3 (2)	3 (2)	3 (4)	3 (2)	3 (2)	**2** (2)

Table 2: Size of transition-based generalized Rabin automata produced by `ltl3dra` or `Rabinizer`, and minimized by our procedure **configured to keep the same acceptance condition.** Acceptance conditions are indicated with "Rabin x" meaning Rabin acceptance with x pairs, "gR x" for generalized-Rabin [5, 3] with x pairs, or "gcB x" for generalized-co-Büchi with x acceptance sets (one of these x sets has to be seen finitely).

	`Rabinizer`		`ltl3dra`	
	orig.	min	orig.	min
$\neg(\overline{p_0} \wedge ((\mathsf{G}\overline{p_0} \wedge ((\mathsf{F}p_0 \wedge \mathsf{G}\overline{p_1}) \vee (\mathsf{G}\overline{p_0} \wedge \mathsf{F}p_1))) \vee (\mathsf{F}p_0 \wedge (\mathsf{G}\overline{p_0} \vee \mathsf{F}p_1))))$	6 (gR 3)	**4**	6 (gcB 3)	**4**
$\neg((\mathsf{XF}p_0 \wedge (p_1 \vee \mathsf{XG}\overline{p_0})) \vee (\mathsf{XG}\overline{p_0} \wedge ((\overline{p_1} \wedge \mathsf{XF}p_0) \vee (p_1 \wedge \mathsf{XG}\overline{p_0}))))$	5 (gR 2)	**3**	5 (gcB 2)	**3**
$\neg(\mathsf{F}p_0 \wedge (p_1 \vee \mathsf{G}p_2) \wedge (\overline{p_1} \vee \mathsf{F}\overline{p_2}))$	8 (gR 4)	8	8 (gcB 3)	8
$\neg(\mathsf{F}p_0 \wedge \mathsf{GF}p_1)$	2 (gcB 2)	2	2 (gcB 2)	2
$\neg(p_0 \vee \mathsf{XG}(p_1 \wedge \mathsf{F}\overline{p_0}))$	4 (gR 2)	4	5 (gcB 2)	**4**
$\overline{p_0} \wedge ((\mathsf{G}\overline{p_0} \wedge ((\mathsf{F}p_0 \wedge \mathsf{G}\overline{p_1}) \vee (\mathsf{G}\overline{p_0} \wedge \mathsf{F}p_1))) \vee (\mathsf{F}p_0 \wedge (\mathsf{G}\overline{p_0} \vee \mathsf{F}p_1)))$	6 (gR 2)	**4**	6 (gcB 2)	**4**
$\mathsf{F}(\mathsf{G}(p_2 \vee (\overline{p_1} \wedge \overline{p_2})) \vee (\mathsf{X}p_1\,\mathsf{U}(p_0 \wedge \mathsf{X}p_1)))$	3 (gR 2)	3	4 (gcB 2)	**3**
$\mathsf{F}p_0 \wedge (p_1 \vee \mathsf{G}p_2) \wedge (\overline{p_1} \vee \mathsf{F}\overline{p_2})$	8 (gR 2)	8	8 (gcB 2)	8
$\mathsf{G}p_0 \wedge \mathsf{XF}p_1$	4 (gcB 1)	4	4 (gcB 1)	4
$\mathsf{F}\overline{p_0} \wedge \mathsf{X}p_0 \wedge (\mathsf{G}p_1 \vee \mathsf{XF}\overline{p_0})$	7 (gR 2)	7	9 (gcB 2)	**7**
$p_0 \vee \mathsf{XG}(p_1 \wedge \mathsf{F}\overline{p_0})$	5 (gR 2)	**4**	4 (gR 3)	4

4 Conclusion

We have presented a tool that can read any deterministic ω-automaton and synthetize (if it exists) an equivalent deterministic ω-automaton with a given number of states and arbitrary acceptance condition.

Although the SAT-based encoding is exponential in the number of acceptance sets, our experience is that it is nonetheless usable for automata that have up to 8 acceptance sets. This is enough to cover a large spectrum of temporal properties.

By processing the output of existing translators, we were able to find several cases where smaller automata exist, showing that there is still room for improvement in tools that translate LTL into ω-automata.

As a final remark, we should point that our tool can find a minimal automaton for a user-supplied acceptance condition. It might make sense to specify a more complex acceptance condition in order to obtain a smaller automaton. Could such a better acceptance condition be synthetized automatically?

References

1. G. Audemard and L. Simon. Predicting learnt clauses quality in modern SAT solvers. In *IJCAI'09*, pp. 399–404, July 2009.
2. S. Baarir and A. Duret-Lutz. Mechanizing the minimization of deterministic generalized Büchi automata. In *FORTE'14*, vol. 8461 of *LNCS*, pp. 266–283. Springer, June 2014.
3. T. Babiak, F. Blahoudek, M. Křetínský, and J. Strejček. Effective translation of LTL to deterministic Rabin automata: Beyond the (F, G)-fragment. In *ATVA'13*, vol. 8172 of *LNCS*, pp. 24–39. Springer, 2013.
4. T. Babiak, F. Blahoudek, A. Duret-Lutz, J. Klein, J. Křetínský, D. Müller, D. Parker, and J. Strejček. The Hanoi Omega-Automata Format. In *CAV'15*, vol. 8172 of *LNCS*, pp. 442–445. Springer, 2015. See also `http://adl.github.io/hoaf/`.
5. K. Chatterjee, A. Gaiser, and J. Křetínský. Automata with generalized Rabin pairs for probabilistic model checking and LTL synthesis. In *CAV'13*, vol. 8044 of *LNCS*, pp. 559–575. Springer, 2013.
6. A. Duret-Lutz. Manipulating LTL formulas using Spot 1.0. In *ATVA'13*, vol. 8172 of *LNCS*, pp. 442–445. Springer, 2013.
7. R. Ehlers. Minimising deterministic Büchi automata precisely using SAT solving. In *SAT'10*, vol. 6175 of *LNCS*, pp. 326–332. Springer, 2010.
8. J. Klein and C. Baier. On-the-fly stuttering in the construction of deterministic ω-automata. In *CIAA'07*, vol. 4783 of *LNCS*, pp. 51–61. Springer, 2007.
9. Z. Komárková and J. Křetínský. Rabinizer 3: Safraless translation of LTL to small deterministic automata. In *ATVA'14*, vol. 8837 of *LNCS*, pp. 235–241. Springer, 2014.
10. J. Křetínský and J. Esparza. Deterministic automata for the (F,G)-fragment of LTL. In *CAV'12*, vol. 7358 of *LNCS*, pp. 7–22. Springer, 2012.
11. J. Křetínský and R. Ledesma-Garza. Rabinizer 2: Small deterministic automata for LTL \ GU. In *ATVA'13*, vol. 8172 of *LNCS*, pp. 446–450. Springer, 2013.
12. M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *CAV'11*, vol. 6806 of *LNCS*, pp. 585–591. Springer, 2011.
13. S. Minato. Fast generation of irredundant sum-of-products forms from binary decision diagrams. In *SASIMI'92*, pp. 64–73, Apr. 1992.
14. S. Safra. *Complexity of Automata on Infinite Objects*. PhD thesis, The Weizmann Institute of Science, Rehovot, Israel, Mar. 1989.