Reducing Acceptance Marks in Emerson-Lei Automata by QBF-solving

Tereza Schwarzová
Faculty of Informatics, Masaryk University, Brno, Czech Republic

Jan Strejček
Faculty of Informatics, Masaryk University, Brno, Czech Republic

Juraj Major
Faculty of Informatics, Masaryk University, Brno, Czech Republic

Abstract

Transition-based Emerson-Lei automata (TELA) became a popular formalism that generalizes many traditional kinds of automata over infinite words including Büchi, co-Büchi, Rabin, Streett, and parity automata. Transitions in a TELA are labelled with acceptance marks and its accepting formula is a positive Boolean combination of terms saying that a particular mark has to be visited infinitely or finitely often. Algorithms processing these automata are often very sensitive to the number of acceptance marks. We introduce a new technique for reducing the number of acceptance marks in TELA with the use of *quantified Boolean formulas (QBF)*. In fact, we present three formula constructions with increasing reduction potential and formula complexity, and thus also decreasing speed of their satisfiability solving. We evaluated our reduction technique on TELA produced by state-of-the-art tools of the libraries Owl and Spot and by the tool `ltl3tela`. The technique reduced some acceptance marks in automata produced by all the tools. On automata with more than one acceptance mark produced by tools Delag and Rabinizer 4 of the Owl library, our technique reduced 29.1% and 38.5% of acceptance marks, respectively.

2012 ACM Subject Classification Theory of computation → Logic; Theory of computation → Automata over infinite objects

Keywords and phrases Emerson-Lei automata, TELA, automata reduction, QBF, telatko

Digital Object Identifier 10.4230/LIPIcs...

Funding This work has been supported by the Czech Science Foundation grant GA19-24397S.

1 Introduction

Automata over infinite words like Büchi, Rabin, Streett, or parity automata play a crucial role in many algorithms related to the concurrency theory. In particular, they are used in specification, verification, analysis, monitoring, and synthesis of various systems with infinite behaviour. In 1987, Emerson and Lei [12] introduced automata over infinite words where acceptance conditions are arbitrary combinations of acceptance primitives saying that a certain set of states should be visited finitely often or infinitely often. In 2015, the same kind of acceptance condition was described in the *Hanoi omega-automata format (HOAF)* [3]. The only difference is that the acceptance primitives talk about finitely or infinitely often visited acceptance marks rather than sets of states. Acceptance marks are placed on transitions and each mark identifies the set of transitions containing this mark. Hence, these automata are called *transition-based Emerson-Lei automata (TELA)* and they generalize many traditional kinds of automata over infinite words including Büchi, co-Büchi, Rabin, Streett, and parity automata.

TELA have attracted a lot of attention during the last five years. Their popularity comes probably from the fact that these automata can often use less states than equivalent automata with simpler acceptance conditions. Further, algorithms handling TELA can automatically

handle all automata with traditional acceptance conditions. TELA can be obtained for example by translating LTL formulas by `ltl2dela` (known as Delag) [16], `ltl3tela` [15], `ltl2dgra` (known as Rabinizer 4) [13], or `ltl2tgba` which is a part of the Spot library [9]. There are also algorithms processing these automata, for example the emptiness check [5] or translation of TELA to parity automata [17, 7].

Algorithms processing TELA are often sensitive to the number of acceptance marks more than to other parts of the automaton. For example, the transformation of TELA to parity automata based on *color appearance record* [17] transforms a TELA with s states and m acceptance marks into a parity automaton with up to $s \cdot m!$ states. Further, the emptiness check [5] is exponential in the number of acceptance marks that appear in acceptance primitives saying that a mark has to be visited finitely often, while it is only polynomial in other measures of the input automaton.

The number of acceptance marks can be algorithmically reduced to one as every TELA can be transformed to an equivalent Büchi automaton (this can be easily done for example by Spot [9]), but this reduction is paid by dramatic changes of state-space: the number of states can increase exponentially in the number of acceptance marks and some important structural properties like determinism can be lost. This motivates our study of a technique reducing the number of acceptance marks without altering the structure of the automaton.

We present such a technique heavily based on *quantified Boolean formulas (QBF)*. For a given TELA and parameters k, n, it produces a QBF formula which is satisfiable if and only if there exists an automaton with the same structure, n acceptance marks, an acceptance formula in disjunctive normal form with k clauses, and the same set of accepting runs as the original automaton. The placement of the marks on transitions and the acceptance formula can be obtained from a model of the formula. Besides this formula, we describe also the construction of two simpler formulas such that their satisfiability implies the existence of an automaton with the same structure, n acceptance marks, and the same set of accepting runs, but not vice versa.

We have implemented our reduction technique in a tool called `telatko`. We show that the tool can reduce acceptance marks in automata produced by Delag [16], Rabinizer 4 [13] (both included in the Owl library), Spot [9] and `ltl3tela` [15]. While the reduction is relatively modest on TELA produced by `ltl3tela` and Spot, it is substantial on automata produced by the tools of the Owl library.

Related results

There is a simple technique [4] reducing the number of acceptance marks in *transition-based genealised Büchi automata (TGBA)* without changing its structure. We are not aware about any existing research aimed at simplification of acceptance formulas of TELA or reduction of the number of its acceptance marks without increasing the number of states. There exists only a SAT-based approach that transforms a deterministic TELA automaton to an equivalent automaton with a given acceptance condition and a given number of states [2] (if such an automaton exists). Further, there are some SAT-based approaches aimed to reduce the number of states of automata over infinite words. More preceisely, there are reductions designed for nondeterministic Büchi automata [11], deterministic Büchi automata [10], and deterministic generalized Büchi automata [1]. Note that these techniques are usually very slow and their authors typically suggest to use them only for specific purposes like looking for cases where some automata construction can be improved.

Casares, Colcombet, and Fijalkow very recently introduced a structure called *alternating cycle decomposition (ACD)* [6] which compactly represents the information about all accepting

and non-accepting automata cycles. We expect that ACD could potentialy be used to reduce the number of acceptance marks or to simplify the acceptance condition. However, such a reduction is not obvious.

Structure of the paper

The next section introduces basic terms used in the paper. Section 3 explains the construction of three mentioned QBF formulas. The reduction algorithm based on these formulas is presented in Section 4. Section 5 describes our tool `telatko` implemening the reduction technique. Experimental results are shown in Section 6. Finally, Section 7 suggests other applications of our QBF-based reduction technique and closes the paper.

2 Preliminaries

In this section we recall the basic terms around TELA and QBF.

▶ **Definition 1** (TELA). *A* transition-based Emerson-Lei automaton (TELA) *is a tuple* $\mathcal{A} = (Q, M, \Sigma, q_I, \delta, \varphi)$, *where*

- Q *is a finite set of* states,
- M *is a finite set of* acceptance marks,
- Σ *is a finite* alphabet,
- $q_I \in Q$ *is an* initial state,
- $\delta \subseteq Q \times \Sigma \times 2^M \times Q$ *is a* transition relation, *and*
- φ *is the* acceptance condition *constructed according to the following abstract syntax equation, where* m *ranges over* M.

$$\varphi ::= \mathit{true} \mid \mathit{false} \mid \mathsf{Inf}\, m \mid \mathsf{Fin}\, m \mid (\varphi \wedge \varphi) \mid (\varphi \vee \varphi)$$

A tuple $t = (p, a, M', q) \in \delta$ is the *transition* leading from state p to state q labelled with a and acceptance marks M'. The set M' is also refered by $mks(t)$. For a set of transitions $T \subseteq \delta$, let $mks(T) = \bigcup_{t \in T} mks(t)$ denote the set of marks that appear on transitions in T.

A *run* π of $\mathcal{A}$ over an infinite word $u = u_0u_1u_2 \ldots \in \Sigma^\omega$ is an infinite sequence of adjacent transitions $\pi = (q_0, u_0, M_0, q_1)(q_1, u_1, M_1, q_2) \ldots \in \delta^\omega$ where $q_0 = q_I$. Let $inf(\pi)$ denote the set of transitions that appear infinitely many times in π. Run π is *accepting* iff $inf(\pi)$ satisfies the formula φ, where a set T of transitions satisfies $\mathsf{Inf}\, m$ iff $m \in mks(T)$ and it satisfies $\mathsf{Fin}\, m$ iff $m \notin mks(T)$. The *language* of automaton $\mathcal{A}$ is the set $L(\mathcal{A}) = \{u \in \Sigma^\omega \mid$ there is an accepting run of $\mathcal{A}$ over $u\}$. Two automata $\mathcal{A}, \mathcal{B}$ are *equivalent* if $L(\mathcal{A}) = L(\mathcal{B})$.

An acceptance formula φ is in *disjunctive normal formal (DNF)* if it is a disjunction of clauses, where each clause is a conjunction of terms of the form $\mathsf{Fin}\, m$ or $\mathsf{Inf}\, m$. Each acceptance formula can be transformed into an equivalent formula in DNF. Formula *false* corresponds to the disjunction of zero clauses and formula *true* corresponds to one clause with zero terms.

A *path* from a state p to a state q is a finite sequence of adjacent transitions of the form $\rho = (q_0, u_0, M_0, q_1)(q_1, u_1, M_1, q_2) \ldots (q_{n-1}, u_{n-1}, M_{n-1}, q_n) \in \delta^+$ such that $p = q_0$ and $q = q_n$. A nonempty set of states $S \subseteq Q$ is called a nontrivial *strongly connected component (SCC)* if for each $p, q \in S$ there is a path from p to q. An SCC S is *maximal* if there is no SCC S' satisfying $S \subsetneq S'$. In the rest of this paper, SCC always refers to a maximal SCC. Given a set of states $S \subseteq Q$, let $\delta_S = \delta \cap (S \times \Sigma \times 2^M \times S)$ denote the set of all transitions between states in S. Further, for each mark $m \in M$, let $\delta_m = \{t \in \delta \mid m \in mks(t)\}$ denote the set of all transitions marked with m. A set of transitions $T \subseteq \delta$ is called a *cycle* is there

exists a path from a state p to the same state containing each transition of T at least once and no transition outside T. Finally, we assume that each TELA $\mathcal{A}$ contains only states q that are reachable from the initial state q_I (i.e., $q = q_I$ or there is a path from q_I to q) as states that are not reachable from q_I can be eliminated without any impact on $L(\mathcal{A})$.

In graphical representation of automata, we often use acceptance marks $\{1\}, \{2\}, \ldots \in M$. Further, an edge $p \xrightarrow{a\ \{j\},\{k\}} q$ denotes the transition $(p, a, \{j, k\}, q) \in \delta$.

Quantified Boolean formulas (QBF) are Boolean formulas extended with universal and existential quantification over propositional variables. We assume that subformulas of the form $\forall x.\psi$ and $\exists x.\psi$ do not contain another quantification of variable x inside ψ. The semantics of $\forall x.\psi$ and $\exists x.\psi$ is given by equivalences

$$\begin{aligned} \forall x.\psi &\equiv \psi[x \to \mathit{true}] \ \wedge\ \psi[x \to \mathit{false}] \\ \exists x.\psi &\equiv \psi[x \to \mathit{true}] \ \vee\ \psi[x \to \mathit{false}] \end{aligned}$$

where $\psi[x \to \rho]$ denotes the formula ψ with all occurrences of x simultaneously replaced by ρ. The equivalences imply that each QBF can be transformed into an equivalent Boolean formula. However, the size of this Boolean formula can be exponential in the size of the original QBF. Let V be the set of all propositional variables. A mapping $\mu : V \to \{\mathit{true}, \mathit{false}\}$ is a *model* of a QBF formula φ iff it is a satisfying assignment of an equivalent Boolean formula. A QBF formula is *satisfiable* iff it has a model.

3 Construction of QBF formulas

Recall that we aim to reduce the number of acceptance marks in a given TELA $\mathcal{A} = (Q, M, \Sigma, q_I, \delta, \varphi)$ without altering its structure and language. In other words, we are looking for a set N of acceptance marks, an accepance formula ψ over N, and a function $nm : \delta \to Q \times \Sigma \times 2^N \times Q$ assigning new marks to transitions (i.e., for each $t = (p, a, M', q) \in \delta$, we assume that $nm(t) = (p, a, N', q)$ for some $N' \subseteq N$) such that $|N| < |M|$ and the automaton $\mathcal{B} = (Q, N, \Sigma, q_I, nm(\delta), \psi)$ is equivalent to $\mathcal{A}$. To ensure that the modified automaton is equivalent to the original one, we actually look for ψ and nm such that each run $\pi = t_0 t_1 t_2 \ldots$ of $\mathcal{A}$ is accepting if and only if the run $nm(t_0)nm(t_1)nm(t_2)\ldots$ of $\mathcal{B}$ is accepting.

We start with several simple observations. Acceptance of a run π is fully determined by the set $\mathit{inf}(\pi)$. Further, each set $\mathit{inf}(\pi)$ for a run π is a cycle and vice versa. Hence, our reduction methods is looking for a new acceptance formula ψ and function nm such that for each cycle $T \subseteq \delta$, it holds that T satisfies φ if and only if $nm(T)$ satisifes ψ. This can be roughly denoted by the formula

$$\forall T \subseteq \delta \ .\ \mathit{cycle}(T) \implies \big(\mathit{satisfies}_{\varphi}(T) \iff \mathit{satisfies}_{\psi,nm}(T)\big).$$

In fact, this corresponds to the shape of the QBF formulas we will construct. As we are looking for ψ and nm such that the formula holds, the subformula $\mathit{satisfies}_{\psi,nm}(T)$ contains many free variables representing possible instances of ψ and nm. If the formula is satisfiable, then each of its models encodes a desired instance of ψ and nm. In the following, we assume that we are looking for a new acceptance formula ψ in DNF. Note that our reduction method can be easily adapted to look for ψ of a different shape.

Now we describe the construction of QBF formulas in detail. The construction is parametrized by two integers $C, K \geq 0$, where K is the desired number of acceptance marks and C is the number of clauses of ψ. For simplicity, we assume that the reduced automaton

will use the acceptance marks $N_K = \{1, 2, \ldots, K\}$. We start with a description of Boolean variables used in the constructed QBF formulas.

- For each transition $t \in \delta$, variable e_t says whether t is in the current set T or not.

$$e_t = \begin{cases} 1 & \text{if } t \in T \\ 0 & \text{otherwise} \end{cases}$$

- For each transition $t \in \delta$ and acceptance mark $k \in N_K$, variable $n_{t,k}$ says whether k is on the transition $nm(t)$ or not.

$$n_{t,k} = \begin{cases} 1 & \text{if } k \in mks(nm(t)) \\ 0 & \text{otherwise} \end{cases}$$

- For each $c \in \{1, 2, \ldots, C\}$ and acceptance mark $k \in N_K$, variables $i_{c,k}$ and $f_{c,k}$ say whether the c^{th} clause of ψ contains terms $\mathsf{Inf}\, k$ or $\mathsf{Fin}\, k$, respectively.

$$i_{c,k} = \begin{cases} 1 & \text{if the } c^{\text{th}} \text{ clause of } \psi \text{ contains } \mathsf{Inf}\, k \\ 0 & \text{otherwise} \end{cases}$$

$$f_{c,k} = \begin{cases} 1 & \text{if the } c^{\text{th}} \text{ clause of } \psi \text{ contains } \mathsf{Fin}\, k \\ 0 & \text{otherwise} \end{cases}$$

By $\vec{e}, \vec{n}, \vec{i}, \vec{f}$ we denote the vectors of all variables of the form e_t, $n_{t,k}$, $i_{c,k}$, and $f_{c,k}$, respectively. The constructed QBF formulas have the form

$$\Phi_{C,K}(\vec{n}, \vec{i}, \vec{f}) \;=\; \forall \vec{e}\,.\; \mathit{cycle}(\vec{e}) \implies \big(\mathit{satisfies}_{\varphi}(\vec{e}) \iff \mathit{satisfies}_{C,K}(\vec{e}, \vec{n}, \vec{i}, \vec{f})\big).$$

It remains to describe the subformulas $\mathit{cycle}(\vec{e})$, $\mathit{satisfies}_{\varphi}(\vec{e})$, and $\mathit{satisfies}_{C,K}(\vec{e}, \vec{n}, \vec{i}, \vec{f})$. The subformula $\mathit{cycle}(\vec{e})$ will be defined as the last one and we present three versions of it.

The subformula $\mathit{satisfies}_{\varphi}(\vec{e})$ says whether T satisfies the original acceptance formula φ and it is derived directly from φ. Recall that T satisfies $\mathsf{Inf}\, m$ iff $m \in mks(T)$, which means that T contains some transition with mark m. As the transitions with mark m form the set δ_m, $\mathsf{Inf}\, m$ can be expressed by $\bigvee_{t \in \delta_m} e_t$. Similarly, T satisfies $\mathsf{Fin}\, m$ iff $m \notin mks(T)$, which can be expressed by $\bigwedge_{t \in \delta_m} \neg e_t$. Hence, $\mathit{satisfies}_{\varphi}(\vec{e})$ arises from φ by replacing

- all terms of the form $\mathsf{Inf}\, m$ by $\bigvee_{t \in \delta_m} e_t$ and
- all terms of the form $\mathsf{Fin}\, m$ by $\bigwedge_{t \in \delta_m} \neg e_t$.

Now we construct the subformula $\mathit{satisfies}_{C,K}(\vec{e}, \vec{n}, \vec{i}, \vec{f})$ that evaluates to true iff $nm(T)$ satisfies ψ. The subformula reflects the basic structure of ψ. As we assume that ψ is a disjunction of C clauses, we have

$$\mathit{satisfies}_{C,K}(\vec{e}, \vec{n}, \vec{i}, \vec{f}) = \bigvee_{c \in \{1,2,\ldots,C\}} \xi_{c,K}(\vec{e}, \vec{n}, \vec{i}, \vec{f})$$

where each $\xi_{c,K}(\vec{e}, \vec{n}, \vec{i}, \vec{f})$ corresponds to one clause. Recall that the presence of terms $\mathsf{Inf}\, k$ and $\mathsf{Fin}\, k$ in the c^{th} clause is given by variables $i_{c,k}$ and $f_{c,k}$, respecively. $\mathsf{Inf}\, k$ is satisfied by $nm(T)$ iff T contains a transition t such that $k \in mks(nm(t))$, which can be expressed as $\bigvee_{t \in \delta}(e_t \wedge n_{t,k})$. Similarly, $\mathsf{Fin}\, k$ is satisfied by $nm(T)$ iff there is no transition $t \in T$ such that $k \in mks(nm(t))$, which can be expressed as $\bigwedge_{t \in \delta} \neg(e_t \wedge n_{t,k})$. Hence, we set

$$\xi_{c,K}(\vec{e}, \vec{n}, \vec{i}, \vec{f}) = \bigwedge_{k \in N_K} \Big(i_{c,k} \implies \bigvee_{t \in \delta}(e_t \wedge n_{t,k})\Big) \wedge \Big(f_{c,k} \implies \bigwedge_{t \in \delta} \neg(e_t \wedge n_{t,k})\Big).$$

It remains to define the subformula $cycle(\vec{e})$. Let $T_{\vec{e}}$ denote the set of transitions represented by $\vec{e}$. The original intended meaning of $cycle(\vec{e})$ is

$$cycle(\vec{e}) \iff T_{\vec{e}} \text{ is a cycle.}$$

In fact, only the direction "$\Longleftarrow$" is needed for the correctness of our reduction method. If there are some valuations of $\vec{e}$ such that $cycle(\vec{e})$ holds and $T_{\vec{e}}$ is not a cycle, then we will superfluously require the equivalence $satisfies_{\varphi}(\vec{e}) \iff satisfies_{C,K}(\vec{e}, \vec{n}, \vec{i}, \vec{f})$ on these valuations. These superfluous constraints can lead to loss of reduction opportunities, but not to incorrectness. This observation allows us to trade the precision of $cycle(\vec{e})$ for its simplicity. formula. We define three versions of $cycle(\vec{e})$:

- $cycle_1(\vec{e})$ is a lightweight version, which only says that $T_{\vec{e}}$ is nonempty. It does not use the information about automaton structure, but it comes with an interesting simplification of the whole formula $\Phi_{C,K}$.
- $cycle_2(\vec{e})$ is an intermediate version. It says that $T_{\vec{e}}$ is nonempty and every transition in $T_{\vec{e}}$ has a preceding and a succeeding transition in $T_{\vec{e}}$, which is a necessary condition for being a cycle, but not a sufficient one.
- $cycle_3(\vec{e})$ is a strict version saying that $T_{\vec{e}}$ is a cycle. Unfortunately, it uses additional universally quantified variables corresponding to automata states. Transformation of $\Phi_{C,K}$ to prenex normal form turns the quantifiers to existential ones and the resulting formula thus contains quantifier alternation.

We write $\Phi_{j,C,K}$ to emphasize that a particular formula $\Phi_{C,K}$ contains the version $cycle_j(\vec{e})$.

3.1 Lightweight version $cycle_1(\vec{e})$

The lightweight version is defined as

$$cycle_1(\vec{e}) = \bigvee_{t \in \delta} e_t$$

which means only that $T_{\vec{e}}$ is nonempty. This condition is satisfied by every cycle.

The formula $\Phi_{1,C,K}$ built with $cycle_1(\vec{e})$ actually says that for every nonempty set $T \subseteq \delta$, T satisfies φ if and only if $nm(T)$ satisfies ψ. Note that the formula is not affected by the placement of transitions in the automaton. The only aspect of a transition t reflected by the formula is its set of marks $mks(t)$. Hence, we do not have to distinguish between transitions with the same sets of marks.

Formally, we define an equivalence $\sim \subseteq \delta \times \delta$ on transitions such that $t_1 \sim t_2$ whenever $mks(t_1) = mks(t_2)$.

▶ **Lemma 2.** *Assume that there is a function nm and a formula ψ such that*

$$\text{for every nonempty set } T \subseteq \delta \text{ it holds } (T \text{ satisfies } \varphi \iff nm(T) \text{ satisfies } \psi). \tag{1}$$

Then there exists a function nm' that respects $\sim$ (i.e., if $t_1 \sim t_2$ then $nm'(t_1) = nm'(t_2)$) and

$$\text{for every nonempty set } T \subseteq \delta \text{ it holds } (T \text{ satisfies } \varphi \iff nm'(T) \text{ satisfies } \psi). \tag{2}$$

Proof. Let nm be a function and ψ a formula such that (1) holds. To construct the function nm', we first select one transition from each equivalence class of $\sim$. For every transition $t \in \delta$, by $\bar{t}$ we denote the selected transition equivalent to t. We define the function nm' as $nm'(t) = nm(\bar{t})$. Clearly, nm' respects the equivalence $\sim$. It remains to show that (2) holds for nm' and ψ.

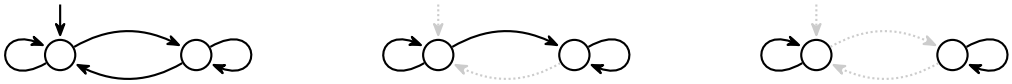


Figure 1 An automaton structure where transition labels and acceptance marks are not depicted (left) and two sets $T_{\vec{e}}$ that are not cycles even if $cycle_2(\vec{e})$ holds (middle and right)

Let $T \subseteq \delta$ be an arbitrary nonempty set. We construct the set $\overline{T} = \{\overline{t} \mid t \in T\}$. As $mks(t) = mks(\overline{t})$ for all transitions, we get $mks(T) = mks(\overline{T})$ and thus

$$T \textit{ satisfies } \varphi \iff \overline{T} \textit{ satisfies } \varphi.$$

Now we apply (1) on $\overline{T}$ to get that

$$\overline{T} \textit{ satisfies } \varphi \iff nm(\overline{T}) \textit{ satisfies } \psi.$$

Finally, the definition of nm' implies that $nm'(T) = nm(\overline{T})$ and thus

$$nm(\overline{T}) \textit{ satisfies } \psi \iff nm'(T) \textit{ satisfies } \psi.$$

Altogether, we obtain

$$T \textit{ satisfies } \varphi \iff \overline{T} \textit{ satisfies } \varphi \iff nm(\overline{T}) \textit{ satisfies } \psi \iff nm'(T) \textit{ satisfies } \psi$$

which proves that (2) holds for nm' and ψ. ◀

The lemma suggests the following simplification of the whole formula $\Phi_{1,C,K}$ built with $cycle_1(\vec{e})$. Before we build the formula, we compute the equivalence $\sim$ on automata transitions and temporarily remove all transitions except one of each equivalence class. Then we build the formula $\Phi_{1,C,K}$ for the pruned automaton. The more transitions we removed, the shorter formula with less e_t variables we obtain. If the formula $\Phi_{1,C,K}$ for the pruned automaton is satisfiable, we derive nm and ψ from its model and extend nm to all transitions of the original automaton such that it changes the acceptance marks on all equivalent transitions in the same way. In the following, we use this simplification whenever $\Phi_{1,C,K}$ is employed.

3.2 Intermediate version $cycle_2(\vec{e})$

The intermediate version says that $T_{\vec{e}}$ is nonempty and for each state $q \in Q$ it holds that $T_{\vec{e}}$ contains a transition leading to q if and only if it contains a transition leading from q. Formally,

$$cycle_2(\vec{e}) = \bigvee_{t \in \delta} e_t \;\wedge\; \bigwedge_{q \in Q} \Big(\bigvee_{t' \in \delta \,\cap\, Q \times \Sigma \times 2^M \times \{q\}} e_{t'} \iff \bigvee_{t'' \in \delta \,\cap\, \{q\} \times \Sigma \times 2^M \times Q} e_{t''} \Big).$$

This condition is satisfied by every cycle, but also by some sets of transitions that are not cycles. Some examples of such sets are provided in Figure 1.

3.3 Strict version $cycle_3(\vec{e})$

Before we give the definition of $cycle_3(\vec{e})$, we prove that cycles can be characterised in the following way.

▶ **Lemma 3.** *A nonempty set $T \subseteq \delta$ is a cycle if and only if for each set of states $S \subseteq Q$ it holds that*

A. *all transitions in T lead from a state in S to a state in S (i.e., $T \subseteq \delta_S$) or*

B. *all transitions in T lead from a state outside S to a state outside S (i.e., $T \subseteq \delta_{Q \setminus S}$) or*

C. *T contains a transition leading from a state in S to a state outside S and a transition leading from a state outside S to a state in S.*

Proof. We first prove the direction "$\Longrightarrow$". Let T be a cycle and $S \subseteq Q$ be an arbitrary set of states. We show that if (A) and (B) do not hold, then (C) has to hold. Hence, assume that $T \not\subseteq \delta_S$ and $T \not\subseteq \delta_{Q \setminus S}$. Then there are two cases.

- T contains a transtition $t \in \delta_S$ and a transition $t' \in \delta_{Q \setminus S}$. The definition of a cycle implies that there exists a path $t_1 t_2 \dots t_n \in T^+$ from a state p back to p containing both t and t'. However, this implies that T contains a transtion leading from a state in S to a state outside S and a transtion leading from a state outside S to a state in S.
- T contains a transition t leading from a state in S to a state outside S (or vice versa). However, as T is a cycle, there exists a path $t_1 t_2 \dots t_n \in T^+$ that leads from a state p to the same state and contains t. Hence, T has to contain also a transition leading from a state outside S to a state in S (or vice versa).

In both cases, (C) holds.

Now we prove the opposite direction "$\Longleftarrow$" by contraposition. Assume that a nonempty set T is not a cycle. We show that there is a set $S \subseteq Q$ such neither (A) nor (B) nor (C) holds. Let p be a state such that some transition of T leads from p. We define the set

$$S_{post} = \{p\} \cup \{q \in Q \mid \text{there is a path in } T^+ \text{ from } p \text{ to } q\}$$

of states reachable from p via transitions in T and the set

$$S_{pre} = \{p\} \cup \{q \in Q \mid \text{there is a path in } T^+ \text{ from } q \text{ to } p\}$$

of states from which p is reachable via transtitions of T. As there is a transition of T leading from p, we have that $T \not\subseteq \delta_{Q \setminus S_{post}}$ and $T \not\subseteq \delta_{Q \setminus S_{pre}}$, i.e., (B) does not holds for S_{post} and S_{pre}. Further, the definition of S_{post} implies that there is no transition of T leading from a state in S_{post} to a state outside S_{post}, which means that (C) does not hold for S_{post}. Similarly, T contains no transition leading from a state outside S_{pre} to a state in S_{pre}, which means that (C) does not hold for S_{pre}. Now we prove by contradition that (A) does not hold for at least one of S_{post}, S_{pre}. Hence, let us assume that $T \subseteq \delta_{S_{post}}$ and $T \subseteq \delta_{S_{pre}}$. Then for each $t_i \in T$ leading from p_i to q_i we have that $p_i \in S_{post}$ and $q_i \in S_{pre}$, which implies that

- $p_i = p$ (we set $\rho'_i = \varepsilon$ in this case) or there is a path $\rho'_i \in T^+$ leading from p to p_i, and
- $q_i = p$ (we set $\rho''_i = \varepsilon$ in this case) or there is a path $\rho''_i \in T^+$ leading from q_i to p.

Then there is a path $\rho_i = \rho'_i t_i \rho''_i \in T^+$ leading from p back to p and containing t_i. If we concatenate all these paths, we get the path $\rho_1 \rho_2 \dots \rho_{|T|} \in T^+$ that contains all transitions of T and leads from p back to p, which means that T is a cycle. This is a contradiction. ◀

The formula $cycle_3(\vec{e})$ says that $T_{\vec{e}}$ is nonempty and each set $S \subseteq Q$ satisfies (A) or (B) or (C). For each state $q \in Q$, variable s_q says whether q is in the current set S or not.

$$s_q = \begin{cases} 1 & \text{if } q \in S \\ 0 & \text{otherwise} \end{cases}$$

By $\vec{s}$ we denote the vectors of all variables of the form s_q. The formula $cycle_3(\vec{e})$ is defined as follows.

$$cycle_3(\vec{e}) = \bigvee_{t \in \delta} e_t \ \wedge \ \forall \vec{s} \,.\, \zeta_A(\vec{e},\vec{s}) \ \vee \ \zeta_B(\vec{e},\vec{s}) \ \vee \ \zeta_C(\vec{e},\vec{s})$$

$$\zeta_A(\vec{e},\vec{s}) = \bigwedge_{t=(p,a,M',q)\in\delta} \big(e_t \implies (s_p \wedge s_q)\big)$$

$$\zeta_B(\vec{e},\vec{s}) = \bigwedge_{t=(p,a,M',q)\in\delta} \big(e_t \implies (\neg s_p \wedge \neg s_q)\big)$$

$$\zeta_C(\vec{e},\vec{s}) = \Big(\bigvee_{t=(p,a,M',q)\in\delta} e_t \wedge s_p \wedge \neg s_q\Big) \ \wedge \ \Big(\bigvee_{t=(p,a,M',q)\in\delta} e_t \wedge \neg s_p \wedge s_q\Big)$$

3.4 SCC-based optimization

Now we present an optimized formula construction based on the fact that every cycle is completely included in the transition set δ_S of some SCC S. Note that we cannot reduce the number of acceptance marks in each SCC separately as all SCCs have to share the same acceptance formula after the reduction. However, for each SCC we can build the formula $\Phi_{C,K}$ separately and then we can put all these formulas into one conjunction such that they share the variables $\vec{i}, \vec{f}$.

Now we describe the construction of optimized QBF formulas $\Phi^{opt}_{j,C,K}$ formally. Given an automaton $\mathcal{A} = (Q, M, \Sigma, \delta, q_I, \varphi)$ and its SCC S, by $\mathcal{A}^S$ we denote the automaton corresponding to the part of $\mathcal{A}$ delimited by S, i.e., $\mathcal{A}^S = (S, M, \Sigma, q', \delta_S, \varphi)$ where q' is an arbitrary state of S. Further, let $\vec{n}_S$ denote the vector of variables $n_{t,k}$ where $t \in \delta_S$ and $k \in N_K$. By $\Phi^S_{j,C,K}(\vec{n}_S, \vec{i}, \vec{f})$ we denote the formula $\Phi_{j,C,K}$ constructed for $\mathcal{A}^S$ as presented above. Finally, we define the optimized QBF formula as

$$\Phi^{opt}_{j,C,K}(\vec{n}', \vec{i}, \vec{f}) = \bigwedge_{\text{SCC } S} \Phi^S_{j,C,K}(\vec{n}_S, \vec{i}, \vec{f})$$

where $\vec{n}'$ is the vector of all varibles contained in vectors $\vec{n}_S$ for SCCs S.

One effect of this optimization is that the formula $\Phi^{opt}_{j,C,K}$ contains no variables $n_{t,k}$ for transitions t that do not lead between states of the same SCC. The acceptance marks on such a transition t do not affect the acceptance of any run as t appears at most once on each run. For these transitions t, we can define $nm(t)$ such that $mks(nm(t)) = \emptyset$.

4 Reduction algorithm

This section explains how we use the QBF formulas constructed in the previous section to reduce the number of acceptance marks in TELA automata. First, we describe a *single-level* reduction, which uses only a single kind of QBF formulas. More precisely, we talk about *level 1*, *level 2*, or *level 3* reduction when $\Phi^{opt}_{1,C,K}$, $\Phi^{opt}_{2,C,K}$, or $\Phi^{opt}_{3,C,K}$ is used, respectively.

The reduction procedure called *SingleLevelReduction* is given in Algorithm 1. Besides the reduction of acceptance marks, the algorithm also reduces the number of clauses in the acceptance formula if the last argument *reduceC* is set to *true*. The first **while** loop gradually decreases the number of marks until $K = 1$ is reached or the QBF solver behind the function *satisfiable*$(\Phi^{opt}_{j,C,K-1})$ fails to reduce the number of marks, i.e., it claims unsatisfiability of the formula or it runs out of resources. If the loop ends with $K = 1$, we check whether an acceptance condition without any mark (i.e., *true* or *false*) can be used. These checks are based on an inspection of the automaton rather than on QBF solving. If some of the checks

Procedure *SingleLevelReduction*$(\mathcal{A}, j, reduceC)$
Input: TELA $\mathcal{A} = (Q, M, \Sigma, \delta, q_I, \varphi)$, level $j \in \{1, 2, 3\}$, $reduceC \in \{true, false\}$
Output: an equivalent TELA with the same structure as $\mathcal{A}$ and with at most as many acceptance marks as in $\mathcal{A}$

$C_{\mathcal{A}} \leftarrow$ the number of clauses in the acceptance formula of $\mathcal{A}$ transformed to DNF
$K_{\mathcal{A}} \leftarrow$ the number of acceptance marks in $\mathcal{A}$
$C \leftarrow C_{\mathcal{A}}$
$K \leftarrow K_{\mathcal{A}}$
while $K > 1 \ \wedge \ satisfiable(\Phi^{opt}_{j,C,K-1})$ **do** $K \leftarrow K-1$
if $K = 1$ **then**
  if *all cycles in* $\mathcal{A}$ *are accepting* **then** `// check the condition` *true*
    return $(Q, \emptyset, \Sigma, \delta', q_I, true)$ where δ' is δ with all marks removed
  if *all cycles in* $\mathcal{A}$ *are rejecting* **then** `// check the condition` *false*
    return $(Q, \emptyset, \Sigma, \delta', q_I, false)$ where δ' is δ with all marks removed
if *reduceC* **then** `// reduction of the number of clauses`
  while $C > 1 \ \wedge \ satisfiable(\Phi^{opt}_{j,C-1,K})$ **do** $C \leftarrow C-1$
if $K < K_{\mathcal{A}} \ \vee \ C < C_{\mathcal{A}}$ **then**
  compute nm and ψ from a model of $\Phi^{opt}_{j,C,K}$
  return $(Q, N_K, \Sigma, q_I, nm(\delta), \psi)$
return $\mathcal{A}$

Algorithm 1 The single-level reduction procedure

succeeds, we return the corresponding automaton without any acceptance mark. Otherwise, if *reduceC* is set to *true* then the procedure gradually reduces the number of clauses in the second **while** loop. Note that the loop never checks for acceptance condition with 0 clauses as it is equivalent to *false* and this case was treated above. Finally, if the procedure succeeds to reduce the number of marks or clauses, it constructs the modified automaton. Otherwise, it returns the original automaton.

The algorithm can be reformulated to use an incremental approach instead of building a new formula in each iteration of the **while** loops. The incremental version of the first **while** loop builds the formula $\Phi = \Phi^{opt}_{j,C,K-1}$ only in the first iteration. In each subsequent iteration, it extends this formula with a condition saying that one more mark is not used in the automaton, i.e., the mark is neither on edges, nor in the acceptance formula. For example, if we want to say that the mark $k \in N_K$ is not used, we replace Φ by

$$\Phi \ \wedge \bigwedge_{t \in \delta} \neg n_{t,k} \ \wedge \bigwedge_{c \in \{1,2,\ldots C\}} (\neg i_{c,k} \wedge \neg f_{c,k}).$$

The second **while** can be transformed to an incremental version similarly. The incremental approach benefits from the fact that some QBF solvers can decide an extended formula faster as they reuse the information computed when solving the original formula.

Figure 2 shows a very simple automaton $\mathcal{A}$ and the three automata produced by calls of *SingleLevelReduction*$(\mathcal{A}, j, true)$ for $j \in \{1, 2, 3\}$. The figure clearly illustrates that the higher level of reduction we use, the more acceptance marks can be reduced. On the other side, lower levels are typically faster. The best results can be often achieved by combining reductions of all levels. We call this approach *multi-level* reduction. It is a straightforward sequential application of the three levels, see Algorithm 2.

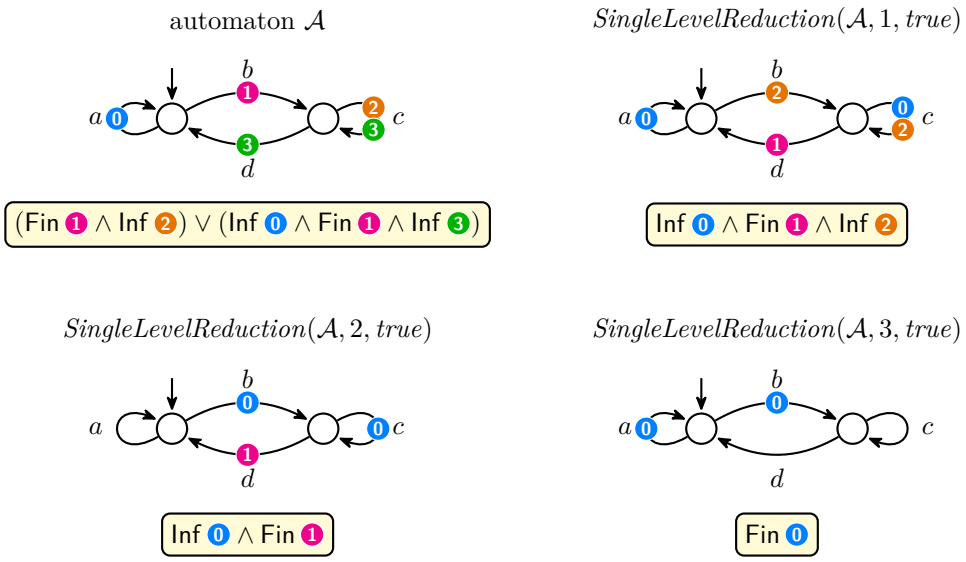


Figure 2 An example illustrating the results of the three single-level reductions: an input automaton $\mathcal{A}$ and the automata obtained by reducing it with level 1, level 2, and level 3.

Procedure *MultiLevelReduction*$(\mathcal{A}, reduceC)$
Input: TELA $\mathcal{A} = (Q, M, \Sigma, \delta, q_I, \varphi)$ and $reduceC \in \{true, false\}$
Output: an equivalent TELA with the same structure as $\mathcal{A}$ and with at most as many acceptance marks as in $\mathcal{A}$

$\mathcal{A} \leftarrow SingleLevelReduction(\mathcal{A}, 1, false)$
$\mathcal{A} \leftarrow SingleLevelReduction(\mathcal{A}, 2, false)$
$\mathcal{A} \leftarrow SingleLevelReduction(\mathcal{A}, 3, reduceC)$
return $\mathcal{A}$

Algorithm 2 The multi-level reduction procedure

5 Implementation

The presented reduction algorithms have been implemented in a tool called `telatko`. The tool is implemented in Python 3 and uses the Spot library [9] for automata parsing and manipulation and the theorem prover Z3 [8] to solve the satisfiability of QBF formulas. `telatko` is available at `https://gitlab.fi.muni.cz/xschwar3/telatko` under the GNU GPLv3 license. The tool can be executed by the command

```
telatko -F <input.hoa> [-L j] [-C] [-I] [-T t] [-O <output.hoa>]
```

where
`-F <input.hoa>` specifies the file with the input automaton in HOA format [3],
`-L` j specifies the reduction level; if omitted, the multi-level reduction is used,
`-C` switches on the reduction of the number of clauses after the number of marks is reduced (it corresponds to $reduceC = true$ in Algortihms 1 and 2),
`-I` switches on the incremental version,
`-T` t sets the timeout for QBF query to t seconds (the default value is 50 seconds),

`-O <output.hoa>` specifies the output file; if omitted, the produced automaton is sent to *stdout* in the HOA format.

If some call of the function $satisfiable(\Phi^{opt}_{j,C,K-1})$ in the first **while** loop of Algorithm 1 does not return *true*, then the name of the output automaton encodes the reason for it. In the case of a single level reduction, the name has the form `L`j`_`k`_`X, where j is the considered level, $k = K - 1$ is the number of acceptance marks considered by the formula, X is either `U` if the formula is unsatisfiable or `T` if the solver did not decided within the time limit. If X is `T`, we longer timeout may lead to further reductions. a If the multi-level reduction is used, the automaton name contains the information from all levels. For example, the name '`L1_5_U L2_3_U L3_1_T`' means that level 1 reduced the number of marks to 6 (reduction to 5 is impossible on this level), level 2 reduced it to 4, and level 3 to 2 as the QBF solver did not finish in the time limit when trying to reduce the number of marks to 1.

6 Experimental evaluation

We have used the tool `genltl` of the Spot library [9] to generate LTL formulas from literature. The parametrized formulas were generated for all combinations of parameter values from 1 to 4. We used the tool `ltlfilt` of the Spot library to simplify the formulas and remove duplicates. Altogether, we have generated 348 unique LTL formulas. These have been translated to deterministic TELA by two state-of-the-art translators from the Owl library [14] and to nondeterministic TELA by another two state-of-the-art translators, namely `ltl2tgba` from the Spot library [9] and `ltl3tela` [15]. Each of the four automata sets produced by individual tools contains less than 348 automata as not all of the formulas are successfully translated due to a timeout of 60 seconds. Further, we have removed automata with 0 or 1 acceptance marks as there is a little point in reducing these. Table 1 shows the exact versions and homepages of used translators, and the number of automata produced by each translator after removing the automata with less than two marks.

For each automata set (that is, the automata produced by an individual LTL translator), we have applied all single-level reductions and the multi-level reduction, always with incremental approach and without reduction of the number of clauses. The timeout for each QBF query was set to 30 seconds. All reductions have been performed by the tool `telatko` build with Spot library version 2.10.4 and Z3 version 4.8.15. The experiments have been run on a computer with Intel® Core™ i7-8700 processor and 32 GB of memory running Ubuntu 20.04.4. We used the tool `autcross` of the Spot library to get the statistics of the reduced automata and the running times.

Table 2 shows the cumulative numbers of marks in the input automata sets and after each reduction, together with the reduction ratio and total time spent by the considered reduction. The column *solver timeout* shows the number of automata for which the last

Table 1 LTL to automata translators used for the experimental evaluation and the number of successfully translated formulas.

tool	version	homepage	automata
`ltl2dela`	21.0	`https://owl.model.in.tum.de/`	129
`ltl2dgra`	21.0		234
`ltl2tgba`	2.10.4	`https://spot.lrde.epita.fr`	70
`ltl3tela`	2.2.0	`https://github.com/jurajmajor/ltl3tela`	91

Table 2 The cumulative numbers of acceptance marks before and after reduction for various reductions and automata sets. The column *reduction* shows the cumulative percentage of saved acceptance marks and *time* reports the cumulative reduction time. The column *solver timeout* indicates the number of instances when the last call to the QBF solver timed out.

automata set	level	marks	reduction [%]	time [s]	solver timeout
`ltl2dela` 129 automata 523 acc. marks	level 1	387	26.0	755.4	18
	level 2	390	25.4	1024.5	20
	level 3	393	24.9	7273.3	25
	multi	371	29.1	9111.4	21
`ltl2dgra` 234 automata 882 acc. marks	level 1	552	37.4	738.7	14
	level 2	569	35.5	907.3	18
	level 3	561	36.4	1163.4	21
	multi	542	38.5	2320.7	21
`ltl2tgba` 70 automata 198 acc. marks	level 1	198	0.0	49.3	0
	level 2	198	0.0	78.9	0
	level 3	189	4.5	1222.8	7
	multi	189	4.5	1259.1	6
`ltl3tela` 91 automata 348 acc. marks	level 1	332	4.6	465.9	12
	level 2	334	4.0	553.9	15
	level 3	326	6.3	691.8	18
	multi	323	7.2	1545.9	18

query to QBF solver did not finished within the 30 seconds limit. The timeout of the last QBF query implies that the automaton may be potentially further reduced by the reduction with a longer time limit.

Table 3 shows the number of automata from all four sets that have a given number of original acceptance marks and a given number of acceptance marks after multi-level reduction. The table indicates that in many cases only 1 or 2 marks can be saved. However, the achieved reduction is substantial for some automata with a higher number of original acceptance marks. For example, in 24 cases, we have reduced 7 or more acceptance marks to only 4.

Figure 3 presents the time spent by multi-level reduction of automata in individual automata sets. The chart shows a pleasing finding that for every set, most automata are reduced in under a second and the high cumulative running times are caused by a few complicated automata.

7 Conclusions

We have presented a method reducing the number of acceptance marks in transition-based Emerson-Lei automata with use of QBF solving and without altering automata structure. We have implemented the method in a tool called `telatko`. The presented experimental results show that the tool can indeed reduce the number of acceptance marks in automata produced by state-of-the-art LTL to automata translators. The reduction was significant on automata produced by the translators of the Owl library.

The reduction of acceptance marks is not the only application of the presented approach. For example, it can be easily adopted to look for an equivalent automaton with the same

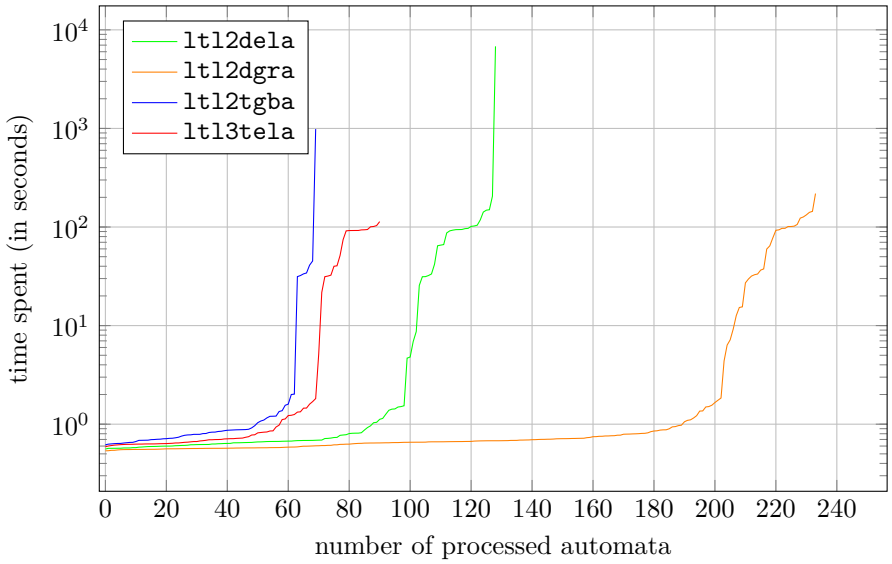
Table 3 The effect of reduction. A cell on coordinates (x, y) contains the number of automata that have been reduced from x to y acceptance marks. If the cell contains a sum of two numbers, the latter represents number of automata where the last call to QBF solver reached the time limit.

acceptance marks after the reduction \ before the reduction	2	3	4	5	6	7	8	9	10–14	15–19	20–24
20–24											**0+1**
15–19										**0+4**	**0+1**
10–14									**0+8**	**0+1**	0
9								**0+1**	0	0	0
8							**0+10**	**0+2**	0	0	0
7						0	**0+2**	0	0	0	0
6					**2+7**	**1+2**	0	0	**0+2**	0	0
5				**7+3**	**2**	0	**0+1**	0	0	0	0
4			**35+7**	**14**	**5+2**	**10**	**4+1**	**4**	**4**	**0+1**	0
3		**51+1**	**5+1**	**2**	0	**1**	0	0	**0+1**	0	0
2	**71+5**	**27**	**9+1**	**4**	**2**	0	**0+1**	0	0	0	0
1	**157**	**11**	**4**	**2**	**1**	**1**	0	0	0	0	0
0	**19**	**2**	**1**	0	0	0	0	0	0	0	0

structure and an acceptance formula of a specific form (e.g., without any $\mathsf{Fin}\,m$ terms). Even though the QBF queries can be time-consuming, in practice one can often find a good trade-off between speed and efficiency by adjusting the formula precision and choosing a reasonable timeout.

In future, we plan to improve the reduction process (e.g., use the bisection method to reduce the number of acceptance marks more efficiently) and study whether the reduction of acceptance condition can enable further reductions or simplifications of automata structures.

Figure 3 Performance of `telatko` on different automata sets. The x axis represents the number of reduced automata, the y axis shows the time spent reducing (in seconds).



References

1. Souheib Baarir and Alexandre Duret-Lutz. Mechanizing the minimization of deterministic generalized Büchi automata. In *Proceedings of the 34th IFIP International Conference on Formal Techniques for Distributed Objects, Components and Systems (FORTE'14)*, volume 8461 of *Lecture Notes in Computer Science*, pages 266–283. Springer, June 2014. doi:10.1007/978-3-662-43613-4_17.
2. Souheib Baarir and Alexandre Duret-Lutz. SAT-based minimization of deterministic ω-automata. In Martin Davis, Ansgar Fehnker, Annabelle McIver, and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning - 20th International Conference, LPAR 2015, Suva, Fiji, November 24-28, 2015, Proceedings*, volume 9450 of *Lecture Notes in Computer Science*, pages 79–87. Springer, 2015. doi:10.1007/978-3-662-48899-7_6.
3. Tomáš Babiak, František Blahoudek, Alexandre Duret-Lutz, Joachim Klein, Jan Křetínský, David Müller, David Parker, and Jan Strejček. The Hanoi Omega-Automata Format. In *Proceedings of the 27th Conference on Computer Aided Verification (CAV'15)*, volume 8172 of *Lecture Notes in Computer Science*, pages 442–445. Springer, 2015. See also http://adl.github.io/hoaf/.
4. Tomáš Babiak, Thomas Badie, Alexandre Duret-Lutz, Mojmír Křetínský, and Jan Strejček. Compositional approach to suspension and other improvements to LTL translation. In Ezio Bartocci and C. R. Ramakrishnan, editors, *Model Checking Software - 20th International Symposium, SPIN 2013, Stony Brook, NY, USA, July 8-9, 2013. Proceedings*, volume 7976 of *Lecture Notes in Computer Science*, pages 81–98. Springer, 2013. URL: https://doi.org/10.1007/978-3-642-39176-7_6.
5. Christel Baier, František Blahoudek, Alexandre Duret-Lutz, Joachim Klein, David Müller, and Jan Strejček. Generic emptiness check for fun and profit. In *Proceedings of the 17th International Symposium on Automated Technology for Verification and Analysis (ATVA'19)*, volume 11781 of *Lecture Notes in Computer Science*, pages 445–461. Springer, 2019. doi:10.1007/978-3-030-31784-3_26.
6. Antonio Casares, Thomas Colcombet, and Nathanaël Fijalkow. Optimal transformations of games and automata using muller conditions. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*, volume 198 of *LIPIcs*, pages 123:1–123:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ICALP.2021.123.
7. Antonio Casares, Alexandre Duret-Lutz, Klara J. Meyer, Florian Renkin, and Salomon Sickert. Practical applications of the Alternating Cycle Decomposition. In *Proceedings of the 28th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, Lecture Notes in Computer Science, 2022. To appear.
8. Leonardo Mendonça de Moura and Nikolaj Bjørner. Z3: an efficient SMT solver. In C. R. Ramakrishnan and Jakob Rehof, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings*, volume 4963 of *Lecture Notes in Computer Science*, pages 337–340. Springer, 2008. doi:10.1007/978-3-540-78800-3_24.
9. Alexandre Duret-Lutz, Alexandre Lewkowicz, Amaury Fauchille, Thibaud Michaud, Etienne Renault, and Laurent Xu. Spot 2.0 — a framework for LTL and ω-automata manipulation. In *Proceedings of the 14th International Symposium on Automated Technology for Verification and Analysis (ATVA'16)*, volume 9938 of *Lecture Notes in Computer Science*, pages 122–129. Springer, 2016. doi:10.1007/978-3-319-46520-3_8.
10. Rüdiger Ehlers. Minimising deterministic Büchi automata precisely using SAT solving. In O. Strichman and S. Szeider, editors, *Proceedings of the 13th Thirteenth International Conference on Theory and Applications of Satisfiability Testing (SAT'10)*, volume 6175 of *Lecture Notes in Computer Science*, pages 326–332. Springer, 2010.

11 Rüdiger Ehlers and Bernd Finkbeiner. On the virtue of patience: Minimizing Büchi automata. In Jaco van de Pol and Michael Weber, editors, *Model Checking Software - 17th International SPIN Workshop, Enschede, The Netherlands, September 27-29, 2010. Proceedings*, volume 6349 of *Lecture Notes in Computer Science*, pages 129–145. Springer, 2010. doi:10.1007/978-3-642-16164-3_10.

12 E. Allen Emerson and Chin-Laung Lei. Modalities for model checking: Branching time logic strikes back. *Science of Computer Programming*, 8(3):275–306, June 1987.

13 Jan Křetínský, Tobias Meggendorfer, Salomon Sickert, and Christopher Ziegler. Rabinizer 4: From LTL to your favourite deterministic automaton. In Hana Chockler and Georg Weissenbacher, editors, *Proceedings of the 30th International Conference on Computer Aided Verification (CAV'18)*, volume 10981 of *Lecture Notes in Computer Science*, pages 567–577. Springer, 2018.

14 Jan Křetínský, Tobias Meggendorfer, and Salomon Sickert. Owl: A library for ω-words, automata, and LTL. In Shuvendu K. Lahiri and Chao Wang, editors, *Automated Technology for Verification and Analysis - 16th International Symposium, ATVA 2018, Los Angeles, CA, USA, October 7-10, 2018, Proceedings*, volume 11138 of *Lecture Notes in Computer Science*, pages 543–550. Springer, 2018. doi:10.1007/978-3-030-01090-4_34.

15 Juraj Major, František Blahoudek, Jan Strejček, Miriama Sasaráková, and Tatiana Zbončáková. ltl3tela: LTL to small deterministic or nondeterministic Emerson-Lei automata. In Yu-Fang Chen, Chih-Hong Cheng, and Javier Esparza, editors, *Automated Technology for Verification and Analysis - 17th International Symposium, ATVA 2019, Taipei, Taiwan, October 28-31, 2019, Proceedings*, volume 11781 of *Lecture Notes in Computer Science*, pages 357–365. Springer, 2019. URL: https://doi.org/10.1007/978-3-030-31784-3_21.

16 David Müller and Salomon Sickert. LTL to deterministic Emerson-Lei automata. In Patricia Bouyer, Andrea Orlandini, and Pierluigi San Pietro, editors, *Proceedings of the Eighth International Symposium on Games, Automata, Logics and Formal Verification (GandALF'17)*, volume 256 of *EPTCS*, pages 180–194, September 2017.

17 Florian Renkin, Alexandre Duret-Lutz, and Adrien Pommellet. Practical "paritizing" of Emerson-Lei automata. In Dang Van Hung and Oleg Sokolsky, editors, *Automated Technology for Verification and Analysis - 18th International Symposium, ATVA 2020, Hanoi, Vietnam, October 19-23, 2020, Proceedings*, volume 12302 of *Lecture Notes in Computer Science*, pages 127–143. Springer, 2020. URL: https://doi.org/10.1007/978-3-030-59152-6_7.