Practical "Paritizing" of Emerson-Lei Automata

Florian Renkin, Alexandre Duret-Lutz, and Adrien Pommellet

LRDE, EPITA, France

Abstract. We introduce a new algorithm that takes a *Transition-based Emerson-Lei Automaton* (TELA), that is, an ω-automaton whose acceptance condition is an arbitrary Boolean formula on sets of transitions to be seen infinitely or finitely often, and converts it into a *Transition-based Parity Automaton* (TPA). To reduce the size of the output TPA, the algorithm combines and optimizes two procedures based on a *latest appearance record* principle, and introduces a *partial degeneralization*. Our motivation is to use this algorithm to improve our LTL synthesis tool, where producing deterministic parity automata is an intermediate step.

1 Introduction

Let us consider the transformation of ω-automata with arbitrary Emerson-Lei acceptance into ω-automata with parity acceptance. Our inputs are *Transition-based Emerson-Lei Automata* (TELA), i.e., automata whose edges are labeled with integer marks like 0, 1, 2, ... and whose acceptance condition is a positive Boolean formula over terms such as Fin(1) or Inf(2) that specifies which marks should be seen infinitely or finitely often in accepting runs. Our algorithm processes a TELA with any such acceptance condition, and outputs a TELA whose acceptance can be interpreted as a *parity max odd* (resp. *even*) condition, i.e., the largest mark seen infinitely often along a run has to be odd (resp. even). Figures 5 and 7 on page 20 show an example of input and output.

While *non-deterministic Büchi automata* are the simplest ω-automata able to represent all ω-regular languages, deterministic Büchi automata are less expressive; as a consequence, applications that require determinism usually switch to more complex acceptance conditions like Rabin, Streett, or parity. Parity can be regarded as the simplest of the three, in the sense that any parity automaton can be converted into a Rabin or a Streett automaton without changing its transition structure. Parity acceptance is especially popular among game solvers, as parity games can be solved with memoryless strategies and arise in many problems.

Our motivation comes from one such problem: *reactive synthesis from LTL specifications*, i.e., building an I/O transducer whose input and output signals satisfy an LTL specification φ [5]. The high-level approach taken by our `ltlsynt` tool [21], or even by the SyntComp'19 winner Strix [19], is to transform the LTL formula into a *deterministic transition-based parity automaton* (DTPA), interpret the DTPA as a parity game by splitting the alphabet on inputs and outputs, then solve the game and use any winning strategy to synthesize a transducer. Let us zoom on the first step: transforming an LTL formula into a DTPA.

One of the many methods to transform an LTL formula into a DTPA is to first convert the LTL formula into a non-deterministic Büchi automaton, and then determinize this automaton using some variant of Safra's construction to obtain a DTPA [23, 24]. This is the current approach of `ltlsynt` [21]. However, since the introduction of the HOA format [3] allowing the representation of TELA, we have seen the development of several tools for converting LTL formulas into TELA: for instance `delag` [22], `ltl2da` and `ltl2na` (all three part of newer versions of Owl [14]), `ltl3tela` [20], or Spot's `ltl2tgba -G` (see Section 6), all trying to reduce the size of their output by using acceptance formulas more closely related to the input LTL formulas. An alternative way to transform an LTL formula into a DTPA is therefore to first transform the LTL formula into a deterministic TELA, and then "paritize" the result. This paper focuses on such a paritization procedure. Note that our construction preserves the deterministic nature of its input but also works on non-deterministic automata.

Our procedure adapts for TELA, optimizes, and combines a few existing transformation procedures. For instance there exists a procedure called SAR (*state appearance record*) [17, 18] that converts a state-based Muller automaton into a state-based parity automaton, and a similar but more specialized procedure called IAR (*index appearance record*) [17, 18] for transforming a Rabin or Streett automaton into a parity automaton. These two procedures are based on a *latest appearance record* (LAR), i.e., a structure that keeps track of the latest occurring state or the latest occurring unsatisfied Rabin/Streett pair (the term LAR is sometimes used to describe SAR [11]). We describe the adaptation of these two procedures in Section 4. In the context of a TELA, we introduce a simplified SAR called CAR (*color appearance record*) that only tracks colors, and the IAR algorithm has already been adapted by Křetínský et al. [16]. A third transformation, also described in Section 4, can be used as a preprocessing before the previous procedures: this is a *partial degeneralization*, i.e. an extension of the classical degeneralization procedure [12, 2] that will replace any sub-formula of the form $\bigwedge_i \mathsf{Inf}(m_i)$ (resp. $\bigvee_i \mathsf{Fin}(m_i)$) by a single $\mathsf{Inf}(m_j)$ (resp. $\mathsf{Fin}(m_j)$) in the acceptance condition.

In Section 5 we present our "paritization" procedure that combines the above procedures with some additional optimizations. Essentially the automaton is processed one strongly-connected component (SCC) at a time, and for each SCC the acceptance condition is simplified before choosing the most appropriate transformation to parity.

This paritization procedure is implemented in Spot 2.9. In Section 6 we show how the combination of all the improvements outperforms the straightforward CAR algorithm in practice.

2 Transition-based Emerson-Lei Automata

Emerson-Lei Automata were defined [9] and named [25] in the 80s; they provide a way to describe a Muller acceptance condition using a positive Boolean formula over sets of states that must be visited finitely or infinitely often. Below we define

the transition-based version of those automata, as used in the *Hanoi Omega-Automata Format* [3]. Instead of working directly with sets of transitions, we label transitions by multiple colored marks, as can be seen in Figures 5–7.

Let $M = \{0, \ldots, n-1\}$ be a finite set of n contiguous integers called the set of *marks* or *colors*, from now on also written $M = \{0, 1, \ldots\}$ in our examples. We define the set $\mathcal{C}(M)$ of *acceptance formulas* according to the following grammar, where m stands for any mark in M:

$$\alpha ::= \top \mid \bot \mid \mathsf{Inf}(m) \mid \mathsf{Fin}(m) \mid (\alpha \wedge \alpha) \mid (\alpha \vee \alpha)$$

Acceptance formulas are interpreted over subsets of M. For $N \subseteq M$ we define the satisfaction relation $N \models \alpha$ inductively according to the following semantics:

$$\begin{array}{lll} N \models \top & N \models \mathsf{Inf}(m) \text{ iff } m \in N & N \models \alpha_1 \wedge \alpha_2 \text{ iff } N \models \alpha_1 \text{ and } N \models \alpha_2 \\ N \not\models \bot & N \models \mathsf{Fin}(m) \text{ iff } m \notin N & N \models \alpha_1 \vee \alpha_2 \text{ iff } N \models \alpha_1 \text{ or } N \models \alpha_2 \end{array}$$

Intuitively, an Emerson-Lei automaton is an ω-automaton labeled by marks whose acceptance condition is expressed as a positive Boolean formula on sets of marks that occur infinitely or finitely often in a run. More formally:

Definition 1 (Transition-based Emerson-Lei Automata). *A* transition-based Emerson-Lei automaton *(TELA) is a tuple* $\mathcal{A} = (Q, M, \Sigma, \delta, q_0, \alpha)$ *where:*

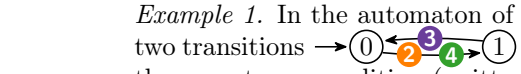
- Q *is a finite set of states.*
- M *is a finite set of marks.*
- Σ *is a finite input alphabet.*
- $\delta \subseteq Q \times \Sigma \times 2^M \times Q$ *is a finite set of transitions.*
- $q_0 \in Q$ *is an initial state.*
- $\alpha \in \mathcal{C}(M)$ *is an acceptance formula.*

Given a transition $d = (q_1, \ell, A, q_2) \in \delta$, we write $d = q_1 \xrightarrow{\ell, A} q_2$. A *run* r of $\mathcal{A}$ is an infinite sequence of transitions $r = (s_i \xrightarrow{\ell_i, A_i} s'_i)_{i \geq 0}$ in δ^ω such that $s_0 = q_0$ and $\forall i \geq 0$, $s'_i = s_{i+1}$. Since Q is finite, for any run r, there exists a position $j_r \geq 0$ such that for each $i \geq j_r$, the transition $s_i \xrightarrow{\ell_i, A_i} s'_i$ occurs infinitely often in r. Let $\mathsf{Rep}(r) = \bigcup_{i \geq j_r} A_i$ be the set of colors **repe**ated infinitely often in r.

A run r is *accepting* if $\mathsf{Rep}(r) \models \alpha$, and we then say that $\mathcal{A}$ *accepts* the word $(\ell_i)_{i \geq 0} \in \Sigma^\omega$. We may then write $r \models \alpha$. The *language* $\mathscr{L}(\mathcal{A})$ is the set of words accepted by $\mathcal{A}$. Two TELA are *equivalent* if they have the same language. By extension, the language of a state $q \in Q$ is the language of the automaton using q as initial state.

Definition 2 (Strongly Connected Component). *Let us consider a TELA* $\mathcal{A} = (Q, M, \Sigma, \delta, q_0, \alpha)$*. A strongly connected component (SCC) is a non-empty set of states* $S \subseteq Q$ *such that any ordered pair of distinct states of* S *can be connected by a sequence of transitions of* δ*.*

We note $\mathcal{A}_{|S} = (S, M, \Sigma, \delta', q'_0, \alpha)$ *a sub-automaton induced by* S*, where* $\delta' = \delta \cap (S \times \Sigma \times 2^M \times S)$*, and* $q'_0 \in S$ *is an arbitrary state of* S*. An SCC* S *is said* accepting *if* $\mathscr{L}(\mathcal{A}_{|S}) \neq \emptyset$*.*

Example 1. In the automaton of Figure 5, the run r that repeats infinitely the two transitions  has $\mathsf{Rep}(r) = \{2, 3, 4\}$. Since $\mathsf{Rep}(r)$ satisfies the acceptance condition (written below the automaton) r is an accepting run. Moreover, the set of states $\{0, 1\}$ is a SCC of the automaton.

A TELA's acceptance formula can be used to express many classical ω-automata acceptance conditions, as shown in Table 1. Note that colors may appear more than once in most formulas; for instance $(\mathsf{Fin}(0) \wedge \mathsf{Inf}(1)) \vee (\mathsf{Fin}(1) \wedge \mathsf{Inf}(0))$ is a Rabin acceptance formula.

Table 1. Shape of classical acceptance formulas. The variables $m, m_0, m_1, \ldots$ stand for any acceptance marks in $M = \{0, 1, \ldots\}$ to allow multiple occurrences.

Büchi	$\mathsf{Inf}(m)$
generalized Büchi	$\bigwedge_i \mathsf{Inf}(m_i)$
co-Büchi	$\mathsf{Fin}(m)$
generalized co-Büchi	$\bigvee_i \mathsf{Fin}(m_i)$
Rabin	$\bigvee_i (\mathsf{Fin}(m_{2i}) \wedge \mathsf{Inf}(m_{2i+1}))$
Rabin-like	$\bigvee_i (\mathsf{Fin}(m_{2i}) \wedge \mathsf{Inf}(m_{2i+1})) \vee \bigvee_j \mathsf{Inf}(m_j) \vee \bigvee_k \mathsf{Fin}(m_k)$
Streett	$\bigwedge_i (\mathsf{Inf}(m_{2i}) \vee \mathsf{Fin}(m_{2i+1}))$
Streett-like	$\bigwedge_i (\mathsf{Inf}(m_{2i}) \vee \mathsf{Fin}(m_{2i+1})) \wedge \bigwedge_j \mathsf{Inf}(m_j) \wedge \bigwedge_k \mathsf{Fin}(m_k)$
parity max even	$\mathsf{Inf}(2k) \vee (\mathsf{Fin}(2k-1) \wedge (\mathsf{Inf}(2k-2) \vee (\mathsf{Fin}(2k-3) \wedge \ldots)))$
parity max odd	$\mathsf{Inf}(2k+1) \vee (\mathsf{Fin}(2k) \wedge (\mathsf{Inf}(2k-1) \vee (\mathsf{Fin}(2k-2) \wedge \ldots)))$

Intuitively, a *Büchi* automaton $\mathcal{A}$ accepts a run r if and only if r visits a given mark infinitely often. If r has instead to visit multiple marks infinitely often, then $\mathcal{A}$ is a *generalized Büchi* automaton. A *co-Büchi* automaton must instead avoid visiting a given mark (or at least one mark amongst a subset of M if it is generalized) infinitely often.

A *Rabin* automaton must visit some marks infinitely often and avoid others, according to at least one pattern amongst a finite set. A *Streett* automaton is the complement of a Rabin automaton. Some acceptance conditions are said to be *Rabin-like* (resp. *Streett-like*) if they can be obtained by removing some Fin or Inf clauses from a Rabin (resp. Streett) acceptance formula. A *parity max even* (resp. *odd*) automaton accepts a run r if and only if the greatest infinitely recurring mark in r is even (resp. odd).

Note that the only unusual formulas in Table 1 are the *Rabin-like* and *Streett-like* conditions. A Rabin-like formula $\bigvee_i \big(\mathsf{Fin}(m_{2i}) \wedge \mathsf{Inf}(m_{2i+1})\big) \vee \bigvee_j \mathsf{Inf}(m_j) \vee \bigvee_k \mathsf{Fin}(m_k)$ can be converted into the Rabin formula $\bigvee_i \big(\mathsf{Fin}(m_{2i}) \wedge \mathsf{Inf}(m_{2i+1})\big) \vee \bigvee_j (\mathsf{Fin}(a) \wedge \mathsf{Inf}(m_j)) \vee \bigvee_k (\mathsf{Fin}(m_k) \wedge \mathsf{Inf}(b))$ by introducing two new marks a and b such that a occurs nowhere in the automaton and b occurs everywhere. Therefore, without loss of generality, we may describe algorithms over Rabin automata, but in practice we implement those over Rabin-like acceptance conditions.

When discussing Rabin acceptance, it is common to mention the number of *Rabin pairs*, i.e., the number of disjuncts in the formula; we use the same

vocabulary for Rabin-like, even if some of the pairs only have one term. Dually, the number of pairs in a Streett-like formula is the number of conjuncts.

Remark 1. Formula $\mathsf{Fin}(0) \wedge \mathsf{Inf}(1)$ can be seen as Rabin with one pair, or a Streett-like with two pairs. Similarly, a generalized Büchi is also Streett-like.

Remark 2. Any sub-formula of the form $\bigvee_i \mathsf{Inf}(m_i)$ (resp. $\bigwedge_i \mathsf{Fin}(m_i)$) can be replaced by a single $\mathsf{Inf}(a)$ (resp. $\mathsf{Fin}(a)$) by introducing a mark a on all transitions where any m_i occurred. Thus, any parity automaton can be rewritten as Rabin-like or Streett-like without adding or removing any transition: to produce a Rabin-like (resp. Streett-like) acceptance, rewrite the parity acceptance formula in disjunctive normal form (resp. CNF) and then replace each term of the form $\bigwedge_i \mathsf{Fin}(m_i)$ (resp. $\bigvee_i \mathsf{Inf}(m_i)$) by a single Fin (resp. Inf).

Remark 3. The following table describes *parity max odd* and *parity max even* acceptance formulas for various number of colors, using the HOA syntax [3]. It may help clarify corner cases for some formulas (e.g. with 0 or 1 color), or provide alternative interpretations as other classical acceptance conditions if there are few enough colors. For Rabin(-like) and Streett(-like), the number of pairs is specified between parentheses (see Remark 1).

	cond.	formula	alt. interpretation
max odd	0	$\top$	accept all
	1	$\mathsf{Fin}(0)$	co-Büchi
	2	$\mathsf{Inf}(1) \vee \mathsf{Fin}(0)$	Streett(1), Rabin-like(2)
	3	$\mathsf{Fin}(2) \wedge (\mathsf{Inf}(1) \vee \mathsf{Fin}(0))$	Streett-like(2)
	4	$\mathsf{Inf}(3) \vee (\mathsf{Fin}(2) \wedge (\mathsf{Inf}(1) \vee \mathsf{Fin}(0)))$	
	5	$\mathsf{Fin}(4) \wedge (\mathsf{Inf}(3) \vee (\mathsf{Fin}(2) \wedge (\mathsf{Inf}(1) \vee \mathsf{Fin}(0))))$	
max even	0	$\bot$	reject all
	1	$\mathsf{Inf}(0)$	Büchi
	2	$\mathsf{Fin}(1) \wedge \mathsf{Inf}(0)$	Rabin(1), Streett-like(2)
	3	$\mathsf{Inf}(2) \vee (\mathsf{Fin}(1) \wedge \mathsf{Inf}(0))$	Rabin-like(2)
	4	$\mathsf{Fin}(3) \wedge (\mathsf{Inf}(2) \vee (\mathsf{Fin}(1) \wedge \mathsf{Inf}(0)))$	
	5	$\mathsf{Inf}(4) \vee (\mathsf{Fin}(3) \wedge (\mathsf{Inf}(2) \vee (\mathsf{Fin}(1) \wedge \mathsf{Inf}(0))))$	

Note that according to the HOA format, a transition can be labeled with any number of colors. Obviously, if the acceptance condition is a *parity max* condition, then transitions with multiple colors can be simplified by removing all colors but the greatest. Moreover, the absence of colors behaves as an imaginary color -1 in terms of *parity max* acceptance. Applications that require each transition to feature exactly one color may simply increment all existing colors, introduce 0 on uncolored transitions, and switch the parity criterium from even to odd or odd to even.

3 Acceptance Simplifications

We introduce in this section straightforward simplification rules that allow us to reduce the number of colors and the size of an acceptance formula of a TELA. Assume that $\mathcal{A}$ is a TELA with an acceptance condition α. The notation $\alpha[\beta \leftarrow \gamma]$ stands for "replace any subformula of α equal to β by γ".

Basic cleanup.
If a color i does not appear on any transition of $\mathcal{A}$, then overwrite α with $\alpha[\mathsf{Fin}(i) \leftarrow \top][\mathsf{Inf}(i) \leftarrow \bot]$.
If a color i appears on all transitions of $\mathcal{A}$, then overwrite α with $\alpha[\mathsf{Fin}(i) \leftarrow \bot][\mathsf{Inf}(i) \leftarrow \top]$.

Merging colors.
If two colors i and j always occur together, overwrite α with $\alpha[\mathsf{Fin}(j) \leftarrow \mathsf{Fin}(i)][\mathsf{Inf}(j) \leftarrow \mathsf{Inf}(i)]$ and remove all occurrences of color j in $\mathcal{A}$.

Simplifying covering colors.
If each transition contains i, j, or both, then α can go through the following four rewriting rules:

$$\alpha[\mathsf{Fin}(i) \wedge \mathsf{Inf}(j) \leftarrow \mathsf{Fin}(i)][\mathsf{Fin}(i) \wedge \mathsf{Fin}(j) \leftarrow \bot]$$
$$[\mathsf{Fin}(i) \vee \mathsf{Inf}(j) \leftarrow \mathsf{Inf}(j)][\mathsf{Inf}(i) \vee \mathsf{Inf}(j) \leftarrow \top]$$

Unit propagation.
$\mathsf{Inf}(i)$ and $\mathsf{Fin}(i)$ behave like positive and negative literals in a formula. Thus, if they appear as unit clauses in a conjunction a or disjunction, then can be propagated to the other clauses. In a subformula of the form $\mathsf{Inf}(i) \vee \beta$ or $\mathsf{Fin}(i) \wedge \beta$, the subformula β can be replaced by $\beta[\mathsf{Inf}(i) \leftarrow \bot][\mathsf{Fin}(i) \leftarrow \top]$. Similarly, in a subformula of the form $\mathsf{Fin}(i) \vee \beta$ or $\mathsf{Inf}(i) \wedge \beta$, the subformula β can be replaced by $\beta[\mathsf{Inf}(i) \leftarrow \top][\mathsf{Fin}(i) \leftarrow \bot]$.

Fusing Inf-disjuncts or Fin-conjuncts.
As per Remark 2, a formula of the form $\mathsf{Inf}(i) \vee \mathsf{Inf}(j)$ (resp. $\mathsf{Fin}(i) \wedge \mathsf{Fin}(j)$) can be replaced by $\mathsf{Inf}(k)$ (resp. $\mathsf{Fin}(k)$) if we add a color k to all transitions with either i and j. As we do not want to increase the number of colors, we only perform such a rewriting when either i or j occurs only once in the formula.
Assuming the colors $m_1, \ldots, m_n$ occur only once in α, the general rule consists in substituting a subformula $\bigwedge_{i=1}^{n}(\mathsf{Inf}(m_i) \vee \beta_i)$ for any subformula $\mathsf{Inf}(j) \vee \bigwedge_{i=1}^{n}(\mathsf{Inf}(m_i) \vee \beta_i)$, and replacing all occurrences of color j in the automaton by the set of colors $m_1, \ldots, m_n$. The rule is self-explanatory if we first distribute $\mathsf{Inf}(j)$ inside the $\bigwedge_{i=1}^{n}$ before applying the rule described in the previous paragraph.
A dual rule allows us to remove $\mathsf{Fin}(j)$ from $\mathsf{Fin}(j) \wedge \bigvee_{i=1}^{n}(\mathsf{Fin}(m_i) \wedge \beta_i)$.

Transition-based acceptation.
It may happen that a TELA has more colors than edges. We can then reduce the number of colors by assigning a single color to each edge then rewriting the acceptance formula. Formally, let us assume the automaton has m edges.

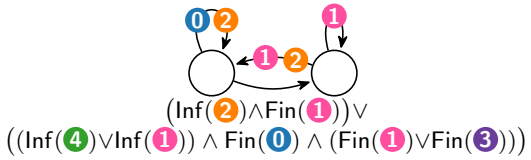


Fig. 1. A first example

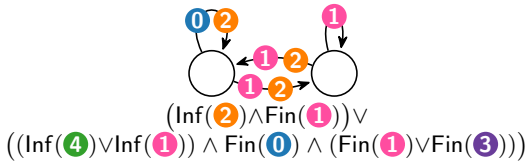


Fig. 2. A second example

We define a bijective coloring function $\kappa : \delta \to \{0, \ldots, m-1\}$. Given $c \in M$, let δ_c be the subset of edges in δ colored by c; we then apply new rewriting rules in order to compute a new acceptance formula:

$$\alpha[\mathsf{Inf}(c) \leftarrow \bigvee_{e \in \delta_c} \mathsf{Inf}(\kappa(e))][\mathsf{Fin}(c) \leftarrow \bigwedge_{e \in \delta_c} \mathsf{Fin}(\kappa(e))]$$

Applying this simplification makes the *Color Acceptance Record* algorithm of Section 4 a *Transition Acceptance Record*.

Basic cleanup, merging colors, and simplifying complementary colors were already implemented in Spot. We added unit propagation and fusing as we worked on the paritization procedure, while looking at the effect of the algorithm on automata with random acceptance conditions.

Example 2. We can see in Figure 3 that neither 3 nor 4 appear in any transition, so $\mathsf{Inf}(4)$ is replaced by $\bot$ and $\mathsf{Fin}(3)$ by $\top$. The condition $(\mathsf{Inf}(2) \wedge \mathsf{Fin}(1)) \vee ((\bot \vee \mathsf{Inf}(1)) \wedge \mathsf{Fin}(0) \wedge (\mathsf{Fin}(1) \vee \top))$ is a Boolean condition that we can simplify and we end up with $(\mathsf{Inf}(2) \wedge \mathsf{Fin}(1)) \vee (\mathsf{Inf}(1) \wedge \mathsf{Fin}(0))$.

Example 3. We consider in Figure 3 a similar example with extra colors. We rewrite the condition again as $(\mathsf{Inf}(2) \wedge \mathsf{Fin}(1)) \vee (\mathsf{Inf}(1) \wedge \mathsf{Fin}(0))$ However, we can also remark that any transition features either 1 or 2. So we can simplify covering colors and get $\mathsf{Fin}(1) \vee (\mathsf{Inf}(1) \wedge \mathsf{Fin}(0))$. This condition is of the form $\mathsf{Fin}(1) \vee \beta$; thus, any occurrence of $\mathsf{Inf}(1)$ is replaced by $\top$ as we apply the unit propagation. We end up with $\mathsf{Fin}(1) \vee \mathsf{Fin}(0)$.

4 Specialized Transformations

We describe three algorithms that transform the acceptance condition of a TELA. The first two output an equivalent TELA with parity acceptance: CAR

(Section 4.1) works for any input, while IAR (Section 4.2) is more suitable for Rabin-like or Streett-like inputs. The third algorithm is a *partial degeneralization* (Section 4.3): given an input automaton with a generic acceptance formula α, it outputs an automaton where any generalized Büchi (resp. generalized co-Büchi) subformula of α has been replaced by a Büchi (resp. co-Büchi) formula. Various optimizations common to these algorithms are listed in Section 4.4.

4.1 Color Appearance Record

Consider a set of marks $M = \{0, 1, \ldots, n-1\}$ and a TELA $\mathcal{A} = (Q, M, \Sigma, \delta, q_0, \alpha)$. We want to compute an equivalent TELA $\mathcal{A}'$ with a *parity max* acceptance condition. Let $M' = \{0, \ldots, 2n+1\}$ be a set of $2n+2$ output marks and $\alpha' = \mathsf{Fin}(2n+1) \wedge (\mathsf{Inf}(2n) \vee (\ldots))$ be the *parity max even* condition on M'.

The intuition behind the *Color Appearance Record* (CAR) algorithm is to pair states in Q with a permutation of M that records the colors visited by a run in the order they were last seen. Colors that are encountered infinitely often will end up being collected to the left of the permutation. Let $\Pi(M)$ be the set of *permutations* (or *histories*) of M, and $\mathsf{Seq}(M)$ be the set of finite injective sequences (or *orderings*) of elements of M.

We can represent a history $\sigma \in \Pi(M)$ by a table $\langle \sigma(0), \sigma(1), \ldots, \sigma(n-1) \rangle$; using the operation $\cdot$ we can concatenate tables. We may occasionally interpret a permutation table as a sequence of colors. We say that a history σ *displays* a set $F \subseteq M$ if there exists an index $i \in M$ such that $F = \{\sigma(0), \ldots, \sigma(i-1)\}$, i.e. F contains exactly the i first elements of σ; we then also say that σ *separates* F at the index i. We introduce the set $Q^{\mathrm{CAR}} = Q \times \Pi(M)$ of CARs over the states in Q.

We introduce an *update* function $\mathcal{U} : \Pi(M) \times \mathsf{Seq}(M) \to \Pi(M) \times M$ that moves a sequence of colors $(c_1, \ldots, c_k)$ to the left of an input permutation table σ and returns the updated history as well as the number of elements rotated (i.e. whose index in the table has been changed) by this move. Unlike the state-based algorithm detailed by Löding et al. [17, 18], a single transition can be labelled by multiple colors. Thus, we may have to insert multiple colors "at the same time" in the history of colors. And the resulting history will depend on the order the colors were inserted in.

Formally, $\mathcal{U}(\sigma, ()) = (\sigma, 0)$ if the input sequence is empty, and otherwise $\mathcal{U}(\sigma, (c_1, \ldots, c_k)) = (\sigma', i)$ where $\sigma' = \langle c_1, \ldots, c_k \rangle \cdot \pi$, the table π is obtained by removing $c_1, \ldots, c_k$ from σ, and $i = 1 + \max(\sigma^{-1}(c_1), \ldots, \sigma^{-1}(c_k))$. Note that σ' is indeed a permutation and that neither the index i nor the table π depend on the ordering of $c_1, \ldots, c_k$. Moreover, the set $\{\sigma(0), \ldots, \sigma(i-1)\}$ is separated by σ' at the index i.

Example 4. Let $n = 4$. Let us assume we want to insert the colors 0 and 1 in the history $\sigma = \langle 2, 1, 0, 3 \rangle$; then $\mathcal{U}(\sigma, (0, 1)) = (\langle 0, 1, 2, 3 \rangle, 3)$ but $\mathcal{U}(\sigma, (1, 0)) = (\langle 1, 0, 2, 3 \rangle, 3)$. Here, $i = 3$ and $\pi = \langle 2, 3 \rangle$.

By $\mathcal{U}$'s definition, the following lemma holds. The first part intuitively means that the last colors inserted are stored to the left of the updated history; the

second, that when we insert a color c, the set of colors that is rotated is exactly the set of colors encountered since c's last occurrence.

Lemma 1. *Let $\sigma \in \Pi(M)$ be a history, $C \subseteq M$, $k \in \mathbb{N}^*$ and $(C_0, \ldots, C_k) \in \mathsf{Seq}(M)^k$ such that $C_0 \cup \ldots \cup C_k = C$. Let $\sigma_0 = \sigma$ and $\mathcal{U}(\sigma_i, C_i) = (\sigma_{i+1}, \iota_{i+1})$ for $i \in \{0, \ldots, k\}$. Then:*

1. *σ_{k+1} displays C.*
2. *If σ displays C, then there exists $l \in \{0, \ldots, k\}$ such that σ_{l+1} separates C at the index ι_{l+1}.*

Proof. Lemma 1.1 is obvious by $\mathcal{U}$'s definition: the latest colors encountered (hence, C) are inserted to the left of the history.

Let us prove Lemma 1.2. For all $c \in C$, let $\mu(c) = \min\{m \mid c \in C_m\}$ be the earliest occurrence of the color c in the sets of labelling marks. Let $l = \max\{\mu(c) \mid c \in C\}$ and $L = \{c \in C \mid \mu(c) = l\}$ be respectively the index and the set of the colors that are encountered last. Note that if we update a history σ displaying C by inserting colors that belong to C, the resulting history will still display C. Hence, for $i \in \{0, \ldots, k+1\}$, σ_i displays C.

Thus, σ_l must be of the form $\pi \cdot \pi_L \cdot \pi'$ where π, π_L, and π' are respectively permutations of $C - L$, L, and $M - C$. Intuitively, the colors in $C - L$ have been moved to the front of σ_l that still displays C. Let $j = |C| - 1$ be the index of the rightmost element of L in σ_l. Obviously, $L \subseteq C_l$. Then by definition of $\mathcal{U}$, $\iota_{l+1} = j + 1$. Since σ_{l+1} displays C, it also separates it at the index ι_{l+1}. □

We intuit that the set of colors occurring infinitely often in a run of $\mathcal{A}$ is exactly the biggest set of colors that are infinitely rotated (i.e. separated at the pivot point) in the CAR. Thus, assuming we match to each separated set in a run a color in M' that depends on the number of elements in the set and will be even if the set verifies the original acceptance condition α and odd otherwise, then we can convert α to a parity max even condition verified by a new run whose states are in Q_{CAR}.

Formally, we introduce a function $\kappa : \Pi(M) \times M \to M'$ defined as follows:

$$\begin{aligned}\kappa(\sigma, i) &= 2i \text{ if } \{\sigma(0), \ldots, \sigma(i-1)\} \models \alpha \\ &= 2i + 1 \text{ otherwise.}\end{aligned}$$

To a given run of of $\mathcal{A}$, we can therefore match several runs that have the same label but also record the colors encountered in a CAR and whose transitions are marked by colors in M':

Definition 3. *Let $r = (s_i \xrightarrow{\ell_i, A_i} s_{i+1})_{i \in \mathbb{N}}$ be a run of $\mathcal{A}$ and $\sigma \in \Pi(M)$. We introduce as follows the set $\rho_{CAR}(r, \sigma) \subseteq (Q^{CAR} \times \Sigma \times 2^{M'} \times Q^{CAR})^\omega$ of* CAR runs *matched to r with initial history σ; $r' = (s'_i \xrightarrow{\ell'_i, B_i} s'_{i+1})_{i \in \mathbb{N}}$ belongs to $\rho_{CAR}(r, \sigma)$ if and only if $\forall i \in \mathbb{N}$:*

- *$\ell_i = \ell'_i$; the two runs have the same label;*

- $s'_i = (s_i, \sigma_i)$; *r' enriches the states of r with a history;*
- $\sigma_0 = \sigma$; *the initial history is equal to the arbitrarily chosen permutation σ;*
- *there exists an ordering $(a_1, \dots, a_k) \in \mathsf{Seq}(M)$ of A_i and an index $\iota_{i+1} \in M$ such that $\mathcal{U}(\sigma_i, (a_1, \dots, a_k)) = (\sigma_{i+1}, \iota_{i+1})$; the history is updated according to the colors encountered by r;*
- $B_i = \{\kappa(\sigma_{i+1}, \iota_{i+1})\}$; *edges are labelled according to the acceptance and the size of the rotated set.*

Note that if each transition of r is labelled by at most one color, then $\rho_{\mathrm{CAR}}(r, \sigma)$ contains exactly one run. However, if at least two colors label one of the transitions, then we can define multiple CAR runs depending on the order we insert colors in the CAR.

Example 5. Let $n = 4$, $\sigma = \langle 2, 1, 0, 3\rangle$, and $\alpha = \mathsf{Fin}(0) \wedge \mathsf{Inf}(1)$. Consider a run r whose first transition is $q \xrightarrow{x,\{0,1\}} q'$. Let σ be the initial CAR. Let us consider the ordering $(0, 1)$; $\mathcal{U}(\sigma, (0, 1)) = (\langle 0, 1, 2, 3\rangle, 3)$, $\kappa(\langle 0, 1, 2\rangle, 3) = 7$ since $\{0, 1, 2, 3\} \not\models \alpha$, thus, we can match the initial transition to a CAR transition $(q, \langle 2, 1, 0, 3\rangle) \xrightarrow{x,\{7\}} (q', \langle 0, 1, 2, 3\rangle)$.

Had we chosen instead the ordering $(1, 0)$, then the resulting CAR transition would have been $(q, \langle 2, 1, 0, 3\rangle) \xrightarrow{x,\{7\}} (q', \langle 1, 0, 2, 3\rangle)$. Thus, to a single run of $\mathcal{A}$, we can match multiple CAR runs, depending on the insertion order of the colors.

We say that r' separates a set F if there exists an index $i \in \mathbb{N}$ such that σ_i separates F at the index ι_i. F is *separated infinitely often* if there exists an infinite number of such indices i. Intuitively, a set is separated by r' if it is exactly the set of elements rotated after inserting some colors in one of the histories of the CAR run. The following lemma intuitively means that the only sets infinitely separated by CAR runs are either Rep or subsets of Rep.

Lemma 2. *Let r be a run of a TELA $\mathcal{A}$ and $r' \in \rho_{CAR}(r)$.*

1. *$\mathsf{Rep}(r)$ is separated infinitely often by r'.*
2. *If r' separates F infinitely often then $F \subseteq \mathsf{Rep}(r)$.*

Proof. We know that there exists a rank i_0 after which the only colors that label the edges of r belong to $\mathsf{Rep}(r)$. Moreover, by definition of Rep, there exists a rank $i_1 \geq i_0$ such that $A_{i_0} \cup \ldots \cup A_{i_1} = \mathsf{Rep}(r)$. Then, by Lemma 1.1 and by definition of ρ_{CAR}, σ_{i_1} displays $\mathsf{Rep}(r)$, regardless of the insertion order of the coloring labels $A_{i_0}, \dots, A_{i_1}$.

By definition of Rep, there also exists a rank $i_2 \geq i_1$ such that $A_{i_1} \cup \ldots \cup A_{i_2} = \mathsf{Rep}(r)$. Then, by Lemma 1.2 and definition of ρ_{CAR}, there exists an index $i_1 \geq u_0 \geq i_2$ such that σ_{u_0} separates $\mathsf{Rep}(r)$ at the index ι_{u_0}. We can more generally compute a strictly monotonic sequence $(u_i)_{i\in\mathbb{N}}$ such that σ_{u_i} separates $\mathsf{Rep}(r)$ at the index ι_i. Hence, Lemma 2.1 holds.

Moreover, as σ_{u_0} separates exactly $\mathsf{Rep}(r)$ at the index ι_{u_0} and the only colors inserted after the rank u_0 belong to $\mathsf{Rep}(r)$, by definition of the update function, $\forall i \geq u_0$, σ_i can only separate a subset of $\mathsf{Rep}(r)$ at the index ι_i. Thus, Lemma 2.2 holds. □

The next lemma proves that $\mathsf{Rep}(r)$ is exactly the biggest set separated by the CAR run r'.

Lemma 3. *Let r be a run of a TELA $\mathcal{A}$, $r' \in \rho_{CAR}(r)$, and $F \subseteq M$. $\mathsf{Rep}(r) = F$ if and only if:*

1. *F is separated infinitely often by r';*
2. *if F' is separated infinitely often by r', either $F' = F$ or $|F| > |F'|$.*

Proof. Let us assume that $\mathsf{Rep}(r) = F$. Proposition 3.1 holds by Lemma 2.1, and 3.2 holds by Lemma 2.2.

Let $F \subseteq M$ be such that propositions 3.1 and 3.2 hold. By Lemma 2.2, $F \subseteq \mathsf{Rep}(r)$. However, by Lemma 2.1, $\mathsf{Rep}(r)$ is separated infinitely often by r'. Thus, either $\mathsf{Rep}(r) = F$ or $|F| > |\mathsf{Rep}(r)|$. Since $F \subseteq \mathsf{Rep}(r)$, necessarily $\mathsf{Rep}(r) = F$. □

The paritization process relies on the following theorem:

Theorem 1. *Let r be a run of a TELA $\mathcal{A}$, $\sigma \in \Pi(M)$ and $r' \in \rho_{CAR}(r, \sigma)$. Then $r \models \alpha$ if and only if $r' \models \alpha'$.*

Proof. By Lemma 3, $\mathsf{Rep}(r)$ is the biggest set separated infinitely often by r'. Thus, if $\mathsf{Rep}(r) \models \alpha$, then the greatest color visited infinitely often by r' is $2\,|\mathsf{Rep}(r)|$, hence, even. And if $\mathsf{Rep}(r) \not\models \alpha$, then the greatest color visited infinitely often by r' is $2\,|\mathsf{Rep}(r)| + 1$, hence, odd. Thus, r is accepting if and only if r' verifies a parity max even condition. □

The last step of this proof is to design a TELA that can generate the runs in $\rho_{\mathrm{CAR}}(r, \sigma)$. Note that the insertion process is intrinsically non-deterministic, yet we want to preserve determinism if the original TELA $\mathcal{A}$ happens to be deterministic. Fortunately, by Theorem 1, it only takes one run in $\rho_{\mathrm{CAR}}(r, \sigma)$ verifying α' to ensure that r is an accepting run. Thus, we only need to enforce the a deterministic insertion order on the colors encountered (that may nonetheless depend on the current state and history).

Formally, an *ordering choice* over the TELA $\mathcal{A}$ is a function $f : \Pi(M) \times \delta \to \mathsf{Seq}(M)$ such that $f(\sigma, q \xrightarrow{x,A} q')$ is an ordering of A. Intuitively, ordering choices are used to determine the order in which colors should be inserted in the CAR, depending on the original history and the transition applied.

Theorem 2. *Let $\sigma_0 \in \Pi(M)$ and f be an ordering choice on $\mathcal{A}$. We introduce the TELA $\mathcal{A}' = (Q^{CAR}, M', \Sigma, \delta^{CAR}, (q_0, \sigma_0), \alpha')$ where δ_{CAR} is defined as follows: for all $d \in \delta$, $d = q \xrightarrow{x,A} q'$, and for all $\sigma \in \Pi(M)$, $d' = (q, \sigma) \xrightarrow{x,B} (q', \sigma')$ belongs to δ_{CAR}, where $\mathcal{U}(\sigma, f(\sigma, d)) = (\sigma', \iota)$ and $B = \{\kappa(\sigma', \iota)\}$.*

Then $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$. Moreover, if $\mathcal{A}$ is deterministic, then $\mathcal{A}'$ is as well.

Proof. By design, to every run r of $\mathcal{A}$, we can match a run r' of $\mathcal{A}'$ such that $r' \in \rho_{\mathrm{CAR}}(r, \sigma_0)$. By Theorem 1, if r is accepting, then so is r'. Hence, $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{A}')$.

Moreover, to each transition in δ_{CAR}, we can match a predecessor in δ. Thus, to each run r' of $\mathcal{A}'$, we can match a run r of A such that $r' \in \rho_{\mathrm{CAR}}(r, \sigma_0)$. By Theorem 1, if r' is accepting, then so is r. Hence, $\mathcal{L}(\mathcal{A}') \subseteq \mathcal{L}(\mathcal{A})$.

Thus, $\mathcal{L}(\mathcal{A}') = \mathcal{L}(\mathcal{A})$. The definition of δ_{CAR} also preserves determinism. □

Example 6. The CAR arrow at the top-right of Figure 6 shows an application of CAR on a small example. Let us ignore the fact that there is no initial state in these "automata" and focus on how transitions of the output (above the arrow) are built from the transitions of the input (below). Assuming we want to build the successors of the output state $(1_1, \langle 0, 2, 1\rangle)$, we look for all successors of input state 1_1. One option is $1_1 \xrightarrow{2} 0_1$. We compute the history $\mathcal{U}(\langle 0, 2, 1\rangle, \{2\})$ of the destination state by moving 2 to the front of the current history, yielding $\langle 2, 0, 1\rangle$. The destination state is therefore $(0_1, \langle 2, 0, 1\rangle)$. The set of colors separated by this insertion is $R = \{0, 2\}$, and since $R \models \alpha$ the resulting transition is labeled by the color in M' $2 \times |R| + 0 = 4$. Another successor is the loop $1_1 \xrightarrow{0}$. In this case, color 0, already at the front of the history, is moved onto itself, so the output is a loop. Since the separated set $R = \{0\}$ is such that $R \not\models \alpha$, the resulting loop is labeled by $2 \times |R| + 1 = 3$.

Note that this construction may produce $|Q| \times n!$ states in the worst case. The initial history σ_0 and the ordering choice may influence the effective size of the resulting automaton, as shown in Section 4.4.

We will prove that this upper bound in $\mathcal{O}(|Q| \times n!)$ is tight. To do so, we need to introduce the set $\mathsf{Rep}(w)$ of letters that appear infinitely often in a word $w \in \Sigma^\omega$. Then:

Theorem 3. *Let $n \in \mathbb{N}$, $n \geq 2$, and $\Sigma = \{1 \dots, n\}$. We consider the language $\mathcal{L}_n = \{(w_i)_{i\in\mathbb{N}} \in \Sigma^\omega \mid \mathsf{Rep}((w_{2i+1})_{i\in\mathbb{N}}) \subseteq \mathsf{Rep}((w_{2i})_{i\in\mathbb{N}})\}$ of infinite words on Σ such that any letter infinitely occurring at odd positions also occurs infinitely at even positions.*

Then any DRA accepting $\mathcal{L}_n$ has at least $2 \times n!$ states.

Consider the set of marks $M_n = \{-n, \dots -1\} \cup \{1 \dots, n\}$ and the DTELA $\mathcal{A}_n = (A, M, \Sigma, \delta_n, a_0, \alpha_n)$, where:

- $A = \{q_+, q_-\}$;
- $a_0 = q_+$;
- $\delta_n = \{q_+ \xrightarrow{i,\{i\}} q_- \mid i \in \Sigma\} \cup \{q_- \xrightarrow{i,\{-i\}} q_+ \mid i \in \Sigma\}$;
- $\alpha_n = \bigwedge_{i=1}^{i=n} \mathsf{Fin}(-i) \vee \mathsf{Inf}(i)$.

Then obviously $\mathcal{L}_n = \mathcal{L}(\mathcal{A}_n)$, as positive (resp. negative) colors are only visited during the even (resp. odd) steps of any run of $\mathcal{A}_n$.

If Theorem 3 holds, then paritizing the automaton $\mathcal{A}_n$ that has two states will result in a parity automaton with at least $2 \times n!$ states, as a parity acceptance condition is merely a special Rabin condition.

Proof. If $n = 2$, consider the automaton $\mathcal{A}_2$ shown in Figure 3 that accepts $\mathcal{L}_2$.

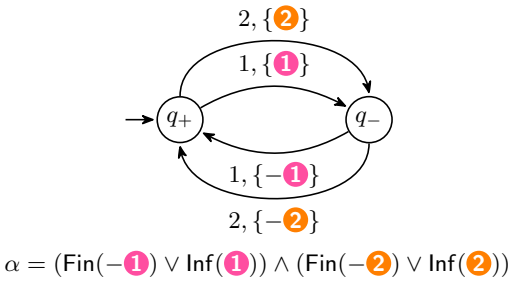


Fig. 3. The automaton $\mathcal{A}_2$.

The automaton $\mathcal{A}_2$ is equivalent to the parity automaton $\mathcal{P}_2$ that features 4 states provided by our IAR implementation (as detailed later in Section 4.2). We can check that this DTELA is minimal in terms of states by applying the algorithm of Baarir et al. [1] implemented in Spot. Thus the theorem holds for $n = 2$.

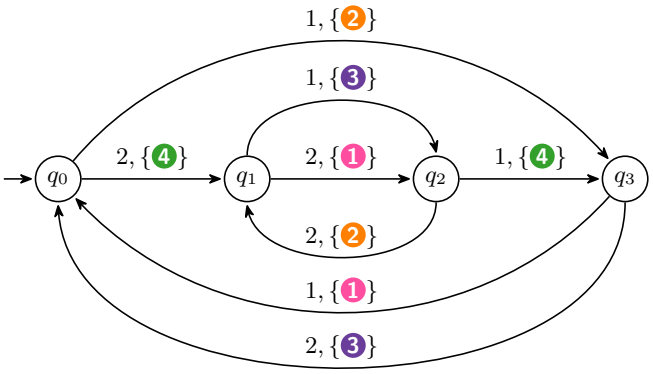


Fig. 4. The automaton $\mathcal{P}_2$.

Let us suppose that the theorem holds for rank $n-1$. We will prove that it holds for rank n. Let $\mathcal{R} = (Q, M, \Sigma, \delta, q_0, \alpha)$ be a DRA accepting $\mathcal{L}_n$. Our goal is to prove that there exist n runs of $\mathcal{R}$ that visit at least $2 \times (n-1)!$ states infinitely often each, and that these sets of states do not overlap.

Let Q_{even} be the set of states of Q reachable from the initial state q_0 using an even number of edges. For $q \in Q_{\text{even}}$, let $\mathcal{R}_q = (Q, M, \Sigma, \delta, q, \alpha)$. Note that $\mathcal{L}_n$

is stable by concatenation with a finite prefix of even length. Thus, $\mathcal{R}_q$ accepts $\mathcal{L}_n$.

Moreover, $\mathcal{L}_{n-1} = \mathcal{L}_n \cap (\Sigma \setminus \{n\})^\omega$. Thus, the restriction of $\mathcal{R}_q$ to the alphabet $\Sigma \setminus \{n\}$ accepts $\mathcal{L}_{n-1}$. In a similar manner, for $i \in \{1, \dots, n\}$, the restriction $\mathcal{R}_q^i$ of $\mathcal{R}_q$ to the alphabet $\Sigma_i = \Sigma \setminus \{i\}$ accepts a language isomorphic to $\mathcal{L}_{n-1}$.

Thus, by induction hypothesis, $\mathcal{R}_q^i$ has at least $2 \times (n-1)!$ states. Actually, it even has a SCC with at least $2 \times (n-1)!$ states: consider a state $q' \in Q_{\text{even}}$ that also belongs to $\mathcal{R}_q^i$'s bottommost SCC. The DRA $\mathcal{R}_{q'}^i$ accepts a language isomorphic to $\mathcal{L}_{n-1}$, and can still do so if we prune every state that isn't in $\mathcal{R}_q^i$'s bottommost SCC. Thus, by applying the induction hypothesis to this restriction of $\mathcal{R}_{q'}^i$, we show that $\mathcal{R}_q^i$'s bottommost SCC has at least $2 \times (n-1)!$ states.

The end of proof then relies on the following lemma:

Lemma 4. *For all $i \in \{1, \dots, n\}$, there exists an infinite word $\alpha^i \in \Sigma_i^\omega$ such that:*

1. *There exists a non-accepting run σ^i of $\mathcal{R}$ labelled by α^i.*
2. *Let us consider the set $\mathsf{Rep}_S(\sigma^i)$ of states visited infinitely often by σ^i. Then $\left|\mathsf{Rep}_S(\sigma^i)\right| \geq 2 \times (n-1)!$.*
3. *For all $j \in \Sigma_i$, $\mathsf{Rep}_S(\sigma^i) \cap \mathsf{Rep}_S(\sigma^j) = \emptyset$.*

Proof. Let $i \in \{1, \dots, n\}$ and q' be a state in Q_{even} that also belongs to $\mathcal{R}_{q_0}^i$'s bottommost SCC. Let $u \in \Sigma_i^*$ be a word leading from q_0 to q' in $\mathcal{R}_{q_0}^i$ (remember that $\mathcal{R}$ is deterministic). We have shown that $\mathcal{R}_{q'}^i$ accepts $\mathcal{L}_{n-1}$. Moreover, there exists a word $w \in \mathcal{L}_{n-1}$ such that $\mathsf{Rep}((w_{2k+1})_{k\in\mathbb{N}}) = \mathsf{Rep}((w_{2k})_{k\in\mathbb{N}}) = \Sigma_i$. Thus, there exists a word $u' \in \Sigma_i^*$ such that $u_0 = u \cdot u'$ is of even length, contains every letter of Σ_i both on even and odd positions, and visits at least $2 \times (n-1)!$ states of $\mathcal{R}_{q_0}^i$.

Let $j \neq i$ and q_1 be the state reached reading $u_0 ij$ in $\mathcal{R}$. In a similar manner to u_0, there exists a word $u_1 \in \Sigma_i^*$ such that u_1 is of even length, contains every letter of Σ_i both on even and odd positions, and visits at least $2 \times (n-1)!$ different states of $\mathcal{R}_{q_1}^i$.

By repeating this procedure, we can design an infinite word $\alpha^i = u_0 i j u_1 i j \dots$ such that $\mathsf{Rep}((\alpha^i_{2k+1})_{k\in\mathbb{N}}) = \Sigma$ and $\mathsf{Rep}((\alpha^i_{2k})_{k\in\mathbb{N}}) = \Sigma_i$. Thus, $\alpha^i \notin \mathcal{L}_n$ and $\mathcal{R}$ rejects α_i. Point 4.1 therefore holds.

Moreover, let σ^i be the run of $\mathcal{R}$ matched to α^i. There exists a rank l such that σ^i only visits states in $\mathsf{Rep}_S(\sigma^i)$ after that rank. However, by construction, the word u_l must visit at least $2 \times (n-1)!$ different states. Thus, $\mathsf{Rep}_S(\sigma^i)$ contains at least $2 \times (n-1)!$ states. Point 4.2 therefore holds.

Let us now suppose that there exist $j \in \Sigma_i$ such that $\mathsf{Rep}_S(\sigma^i) \cap \mathsf{Rep}_S(\sigma^j) \neq \emptyset$. Let q be a state belonging to this intersection. The infinite run σ^i of $\mathcal{R}$ visits q infinitely often, hence we can split it in a infinite sequence $(\sigma^i_k)_{k\in\mathbb{N}}$ of finite runs of $\mathcal{R}$ such that σ^i_0 leads from q_0 to q and for all $k \geq 1$, the non-trivial run σ^i_k leads from q to q and visits every color in $\mathsf{Rep}(\sigma^i)$. We can similarly split σ^j into a infinite sequence of sub-runs $(\sigma^j_k)_{k\in\mathbb{N}}$ leading from q to q and visiting every color in $\mathsf{Rep}(\sigma^j)$.

Now consider the run $\sigma = \sigma_0^i\sigma_1^j\sigma_1^i\sigma_2^j\sigma_2^i\dots$ of $\mathcal{R}$. It must be a non-accepting run of $\mathcal{R}$, as $\mathsf{Rep}(\sigma) = \mathsf{Rep}(\sigma^i) \cup \mathsf{Rep}(\sigma^j)$ and the union of two non-accepting cycles in a Rabin Automata must obviously be non-accepting by definition of the Rabin acceptance condition (see Table 1).

However, let $\alpha = (\alpha_k)_{k\in\mathbb{N}}$ be the label of the run σ. By design of σ, we have $\mathsf{Rep}((\alpha_{2k+1})_{k\in\mathbb{N}}) = \Sigma$, as it is either equal to $\Sigma \cup \Sigma$, $\Sigma_i \cup \Sigma$, $\Sigma \cup \Sigma_j$, or $\Sigma_i \cup \Sigma_j$, depending on whether σ preserves the parity of the positions of the letters infinitely occurring in α^i and α^j. In a similar manner, $\mathsf{Rep}((\alpha_{2k})_{k\in\mathbb{N}}) = \Sigma$. Thus, $\alpha \in \mathcal{L}_n$. This is not possible, as σ is not an accepting run of $\mathcal{R}$.

As a consequence, $\mathsf{Rep}_S(\sigma^i) \cap \mathsf{Rep}_S(\sigma^j) = \emptyset$ and point 4.3 therefore holds. □

Thus $\bigcup_{i=1}^{i=n} \mathsf{Rep}_S(\sigma^i) \subseteq Q$ and $\sum_{i=1}^{i=n} \left|\mathsf{Rep}_S(\sigma^i)\right| = n \times 2 \times (n-1)!$. Hence, $|Q| \geq 2 \times n!$. □

4.2 Index Appearance Record

While CAR can be used to transform Rabin or Streett automata into parity automata, there exists an algorithm more suitable for these subclasses of TELA. Let $\mathcal{A} = (Q, M, \Sigma, \delta, q_0, \alpha)$ be a TELA with a Rabin acceptance condition $\alpha = \bigvee_{i\in\mathcal{I}} (\mathsf{Fin}(p_i) \wedge \mathsf{Inf}(r_i))$. We call (p_i, r_i) a *Rabin pair*, where p_i is the *prohibited* color, and r_i the *required* color. The intuition here is to track satisfiable Rabin pairs instead of colors. Our goal is to build an automaton $\mathcal{A}'$ with a *parity max odd* α' acceptance over the colors $\mathcal{I}' = \{0, 1, \dots, 2 \times |\mathcal{I}|\}$ that is equivalent to $\mathcal{A}$.

Consider the set $\Pi(\mathcal{I})$ of permutations of Rabin pair indices. We pair states in Q with such a history in order to track the indices of the Rabin pairs (p_i, r_i) in the order the colors p_i were last seen. Let $Q^{\mathrm{IAR}} = Q \times \Pi(\mathcal{I})$ be the set of *index appearance records* (IAR). We update these IAR by using a function $\mathcal{U} : \Pi(\mathcal{I}) \times \mathsf{Seq}(\mathcal{I}) \to \Pi(\mathcal{I})$ similar to the update function in Section 4.1 that rotates a set of indices in front of a history, although $\mathcal{U}$ no longer needs to output the number of elements rotated.

Given a set C of colors, we will also define the set $P(C) = \{i \in \mathcal{I} \mid p_i \in C\}$ of indices of Rabin pairs with a prohibited color in C and the greatest index $\mathcal{M}(\sigma, C) = \max\big(\{-1\} \cup \big\{i \in \{0, \dots, |\mathcal{I}| - 1\} \;\big|\; p_{\sigma(i)} \in C \vee r_{\sigma(i)} \in C\big\}\big)$ in a history σ (as in, furthest to the right) corresponding to a Rabin pair with a color in C, or -1 if there is none.

Intuitively, the prohibited colors of the Rabin pairs whose indices are rotated infinitely often to the left of the history during a run are also encountered infinitely often. Let us consider the pivot point $i = \max\big(\{-1\} \cup \big\{i \in \{0, \dots, |\mathcal{I}| - 1\} \;\big|\; p_{\sigma(i)} \in C\big\}\big)$ after inserting the Rabin indices matched to a set of colors C labelling a transition of a run in a history σ. The Rabin pairs whose index is in $\{\sigma(0), \dots, \sigma(i)\}$ will 'yield' a false result.

Consider then the index $m = \mathcal{M}(\sigma, C)$ of the Rabin pair intersecting C that is the furthest to the right of the permutation. By definition, $m \geq i$. If $m > i$, then $p_{\sigma(m)} \notin C$ and the Rabin pair of index $\sigma(m)$ can be considered 'accepting' for the current transition at least since we encounter its required color while avoiding its prohibited one. Since our goal is to build an equivalent automaton

$\mathcal{A}'$ with *parity max odd* acceptance, we may then design a transition labelled by an 'accepting' odd color whose size is proportional to m. However, if it is not the case, either because C does not intersect a Rabin pair or because $p_{\sigma(m)} \in C$, then we output an even color.

An even color can be 'cancelled' by a greater odd color later in a run: it means that, after encountering the required color of a given pair, we eventually met its associated prohibited color, thus invalidating our initial assessment. Formally, we introduce a coloring function $\kappa : \Pi(\mathcal{I}) \times 2^M \to \mathcal{I}'$ such that:

$$\begin{aligned}\kappa(\sigma, C) &= 0 \text{ if } m = -1\\ &= 2m+1 \text{ if } p_{\sigma(m)} \notin C\\ &= 2m+2 \text{ if } p_{\sigma(m)} \in C\end{aligned}$$

where $m = \mathcal{M}(\sigma, C)$. Moreover, note that the insertion order of the Rabin pairs in the IAR is arbitrary. In order to preserve determinism, we must therefore define an deterministic insertion choice. Formally, a *Rabin ordering choice* over the TELA $\mathcal{A}$ is a function $f : \Pi(\mathcal{I}) \times \delta \to \mathsf{Seq}(\mathcal{I})$ such that $f(\sigma, q \xrightarrow{x,C} q')$ is an ordering of the set $\left\{i \in \{0, \dots, |\mathcal{I}| - 1\} \;\middle|\; p_{\sigma(i)} \in C\right\}$ of Rabin pairs whose forbidden color is in C. At last, we can compute the automaton $\mathcal{A}$':

Theorem 4 ([16]). *Let $\sigma_0 \in \Pi(\mathcal{I})$ and f be a Rabin ordering choice on $\mathcal{A}$. We introduce the TELA $\mathcal{A}' = (Q^{IAR}, \mathcal{I}', \Sigma, \delta^{IAR}, (q_0, \sigma_0), \alpha')$ where δ_{IAR} is defined as follows: for all $d \in \delta$, $d = q \xrightarrow{x,A} q'$, and for all $\sigma \in \Pi(\mathcal{I})$, $d' = (q, \sigma) \xrightarrow{x,B} (q', \sigma')$ belongs to δ_{IAR}, where $\mathcal{U}(\sigma, f(\sigma, d)) = \sigma'$ and $B = \{\kappa(\sigma', A)\}$.*

Then $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$. Moreover, if $\mathcal{A}$ is deterministic, then $\mathcal{A}'$ is as well.

A dual construction transforms Streett into parity max even.

For a proof of this theorem, we refer the reader to Löding [17] (for state-based acceptance) and to Křetínský et al. [16] (for an adaptation to TELA).

This procedure generates an automaton with $|Q| \times |\mathcal{I}|!$ states in the worst case, but unless many colors occur multiple times in α, we usually have $|\mathcal{I}| \leq n/2$, making IAR a more efficient choice than CAR whenever possible. We can always ensure that the number of Rabin pairs is smaller than the number of colors in the automaton, relabelling some edges without changing the automaton's language if needed. The IAR automaton is therefore smaller than the equivalent CAR automaton.

Example 7. The arrow IAR in Figure 6 shows an example of IAR at work on a Rabin automaton with two pairs. The output transition $\boxed{2\langle 01\rangle} \xrightarrow{4} \boxed{2\langle 10\rangle}$ corresponds to a loop labeled by $C = \{0, 2\}$ in the input. Since 0 is prohibited in Rabin pair 1, index 1 has to move to the front of the history. Furthermore, the rightmost index of $\langle 01\rangle$ with a color in C is also $m = 1$ and corresponds to $p_1 = 0 \in C$, thus the output transition is labeled by $2m + 2 = 4$.

Extended IAR: we can trivially extend the transition-based IAR algorithm to Rabin-like and Streett-like automata. Intuitively, an acceptance formula is said to

be Rabin-like (resp. Streett-like) if it can be derived from a Rabin (resp. Streett) acceptance formula by removing either the prohibited Fin or the required Inf from some Rabin (resp. Streett) pairs. Note that a formula can be both Rabin-like and Streett-like, e.g. $\mathsf{Inf}(0) \vee \mathsf{Fin}(1)$.

4.3 Partial Degeneralization

Before talking about the partial degeneralization, we have to describe the degeneralization. A classical degeneralization transforms a generalized Büchi automaton $\mathcal{A}$ (transition-based or state-based) with n states and m colors, into a state-based Büchi automaton with at most $n(m+1)$ states. The general principle is to fix an order of the m colors, duplicate the structure of $\mathcal{A}$ in $m+1$ copies (called levels $0, \ldots, m$), jump from level i to level $i+1$ whenever the color of rank i is encountered, mark the states in the last level as Büchi accepting, and have their outgoing transitions always jump back to level 0 (or better, behave as if they were starting in level 0).

A common optimization of the degeneralization procedure [12, 2] is to let transitions labeled by multiple consecutive colors of D skip several levels at a time. Let us define this *skipping* of levels more formally; we introduce a function $\mathcal{S} : L \times 2^M \to L \times 2^{\{e\}}$ that takes a level i and a set C of colors (labelling some transition), and returns the new level j and a subset that is either $\emptyset$ or $\{e\}$ depending on whether the new color should be added to the output transition or not.

$$\mathcal{S}(i, C) = \begin{cases} (j, \emptyset) & \text{if } j < |D| \\ (j - |D|, \{e\}) & \text{if } j \geq |D| \end{cases}$$

where $j = \max\big(k \in \{i, \ldots, i+|D|\} \big| \{d_{i \bmod |D|}, \ldots, d_{(k+|D|-1) \bmod |D|}\} \subseteq C\big)$ is the size of the longest sequence of consecutive colors of D starting from d_i that can be found in C.

When applying this principle to produce transition-based Büchi automata, only m levels are necessary, and when a transition starting level m sees the color of rank m, it produces an accepting transition going back to level 0. One subtle change made in the definition of $\mathcal{S}(s, C)$ is that those accepting transitions do not always go to level 0, but to level $j - |D|$, i.e., they may also skip levels. To our knowledge, this is the first time this is mentioned.

Let us introduce the partial degeneralization algorithm. Given a TELA A and a subset D of its colors, our intent is to modify A in such a way that we can replace any sub-formula of the form $\bigwedge_{d \in D} \mathsf{Inf}(d)$ in its acceptance condition α by a single $\mathsf{Inf}(e)$ for some new color e. Similarly, any sub-formula of the form $\bigvee_{d \in D} \mathsf{Fin}(d)$ will be replaced by $\mathsf{Fin}(e)$. We denote such a substitution of sub-formulas by $\alpha[\bigwedge_{d \in D} \mathsf{Inf}(d) \leftarrow \mathsf{Inf}(e)][\bigvee_{d \in D} \mathsf{Fin}(d) \leftarrow \mathsf{Fin}(e)]$.

Intuitively, we want to ensure that the runs of the new automaton that see all colors of D infinitely often also see e infinitely often. To do so, we consider once again an ordering $(d_0, d_1, \ldots d_{|D|-1})$ of D and pair each state of the output automaton with a *level* in $L = \{0, 1, \ldots |D| - 1\}$. We jump from a level i to the next level $i+1$ whenever we use a transition labeled by d_i; thus, we reach a level

i only after having met the i first colors of D. We jump back down to level 0 after using a transition t labeled by $d_{|D|-1}$ that leaves a state at level $|D|-1$. Since any cycle going through t has seen the whole set of colors D, we can add the new color e to t.

Our partial degeneralization is simply a generalization of this degeneralization principle to work on any subset of colors in the automaton, regardless of the acceptance condition. In order to do so, we have to keep the original colors in the output, and introduce a new one to mark the points where all tracked colors have been seen. However when applied to subset of colors that appear only once in the acceptance condition, and in a subformula α that is generalized-Büchi or generalized-co-Büchi, then this subformula may be simplified and the original colors discarded.

Theorem 5. *Let $\mathcal{A} = (Q, M, \Sigma, \delta, q_0, \alpha)$ be a TELA, let $D = (d_0, d_1, \ldots, d_{|C|-1})$ be an ordered set of marks of M, and let $L = \{0, 1, \ldots, |D|-1\}$ be a set of* levels.

Let the automaton $\mathcal{A}' = (Q', M', \Sigma, \delta', (q_0, i_0), \alpha')$ be a partial degeneralization of $\mathcal{A}$ according to D, where $Q' = Q \times L$, $M' = M \cup \{e\}$ for some new color $e \notin M$, $\alpha' = \alpha[\bigwedge_{d\in D} \mathsf{Inf}(d) \leftarrow \mathsf{Inf}(e)][\bigvee_{d\in D} \mathsf{Fin}(d) \leftarrow \mathsf{Fin}(e)]$, and $\delta' = \left\{(q_1, i) \xrightarrow{\ell, C} (q_2, j) \;\middle|\; q_1 \xrightarrow{\ell, C\cap M} q_2 \in \delta,\, \mathcal{S}(i, C \cap M) = (j, C \setminus M)\right\}$. The initial level can be any $i_0 \in L$.

Then $\mathcal{A}$ and $\mathcal{A}'$ are equivalent.

First, note that this procedure does not remove any color from the automaton but instead add one: even though subformulas of the form $\bigwedge_{d\in D} \mathsf{Inf}(d)$ are removed from α, other parts of α, preserved in α', may still use colors in D. Of course, colors that do not appear in α' may be removed from the automaton during a subsequent step, as performed by our implementation.

Moreover, since the algorithm preserves all the colors of M, the construction is valid for any subset $D \subseteq M$, even one that does not correspond to a conjunction of Inf or disjunction of Fin in α. In such a case, the construction enlarges the automaton without changing its acceptance condition.

When applied to a generalized-Büchi automaton, and after removing the unused colors, our partial degeneralization produces an automaton similar to what would be produced by any classical degeneralization to transition-based Büchi (modulo the small improvement to $\mathcal{S}(s, C)$ discussed before).

The conversion of generalized-Rabin with k "generalized pairs" into to Rabin automaton with k pairs, presented by Křetínský et al. [13] can be seen as k partial degeneralization run in parallel: each state therefore keeps tracks of a vector of k levels. The same effect can be obtained by running the partial generalization k times, once for each generalized Rabin pair.

For these reasons, we consider that the presented partial degeneralization is a useful building block: it is a single algorithm that can replace existing more specialized techniques (degeneralization of generalized Büchi automata, degeneralization of generalized-Rabin automata).

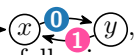
Example 8. In Figure 6, the arrow $\mathrm{PD}_{\{1,3\}}$ denotes the application of a partial degeneralization according to the set $M = (\textcircled{1}, \textcircled{3})$. This allows to rewrite accep-

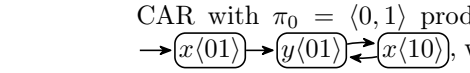
tance's sub-formula $\mathsf{Fin}(1) \vee \mathsf{Fin}(3)$ as $\mathsf{Fin}(4)$ with a new color. Output states (q, i) are written as q_i for brevity. The ordering of colors is $d_0 = 3$, $d_1 = 1$.

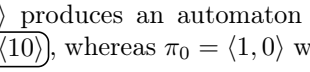
4.4 Optimizations

We now describe several optimizations for the aforementioned constructions.

Jump to bottom: Note that in Theorem 2, the choice of the initial CAR σ_0 is arbitrary, as the DPA we build is always equivalent to the initial TELA; this is true of the initial history of IAR and the initial level i_0 of the partial degeneralization algorithm as well. An improper initial history may lead to a cycle being turned into a lasso.

For instance, if we consider the input automaton 

, applying CAR with $\pi_0 = \langle 0, 1\rangle$ produces an automaton with the following structure: 

, whereas $\pi_0 = \langle 1, 0\rangle$ would yield 

.

Instead of guessing the correct initialization, we simply use the fact that two states (q, σ) and (q, π) recognize the same language: after the algorithm's execution, we redirect any transition leading to a state (q, σ) to the copy (q, π) that lies in the bottommost SCC (according to some topological ordering of the SCCs). The initial state is changed similarly. The input and output automata should have then the same number of SCCs.

This optimization applies to CAR, IAR, partial degeneralization, or combinations of those. E.g., if partial degeneralization is used before CAR or IAR, leading to states of the form $((q, i), \sigma)$, the search for an equivalent state in the bottom SCC needs only consider q, and can simplify both constructions at once.

A similar simplification was initially proposed in the context of IAR for simplifying one SCC at a time [16]. Heuristics used in degeneralization algorithms to select initial level upon entering a new SCC [2] are then unnecessary.

History reuse: Note again that the choice of the ordering function used to process input transitions labeled with multiple colors is arbitrary in Theorem 2. The insertion order of Rabin pair indices in front of the history during an update of the IAR is also arbitrary. Křetínský et al. [16] suggested to check previously built states for one with a compatible trail of the history, in order to avoid creating new states. While implementing this optimization, we noticed that sometimes we can find multiple compatible states: heuristically selecting the most recently created one (as opposed to the oldest one) produces fewer states on average in our benchmark. It seems to create tighter loops and larger "lasso prefixes" that can later be removed by the *jump to bottom* optimization. Such history reuse can also be done *a posteriori* once a candidate automaton has been built, to select better connections.

Heuristic selection of move order: When an input transition is labeled with multiple colors, but no compatible destination state already exists to apply the previous optimization, we select the order in which colors are moved to the front of the history using a heuristic. Colors that are common to all incoming transitions of the destination states are moved last, so they end up at the beginning of the history. For instance in the CAR construction of Figure 6, this is how

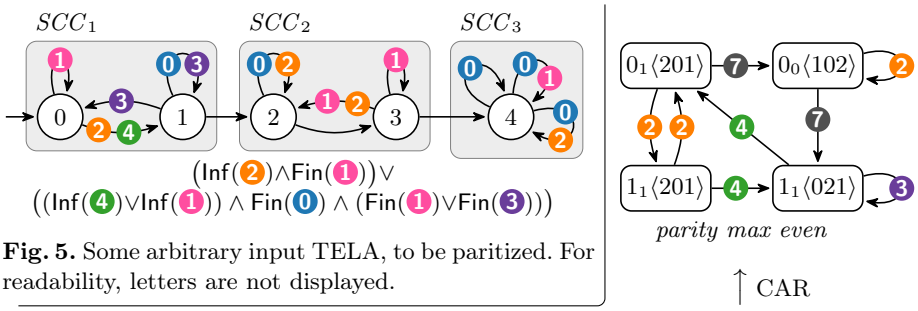


Fig. 5. Some arbitrary input TELA, to be paritized. For readability, letters are not displayed.

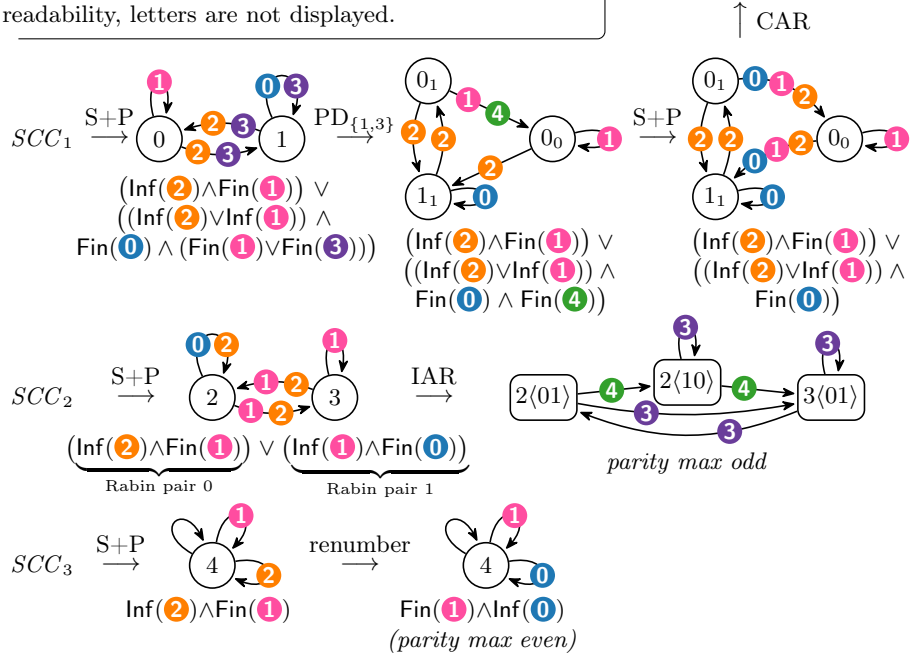


Fig. 6. Intermediate steps of the construction, handling the SCCs in different ways. These steps are explained at various places through Sections 4 and 5.

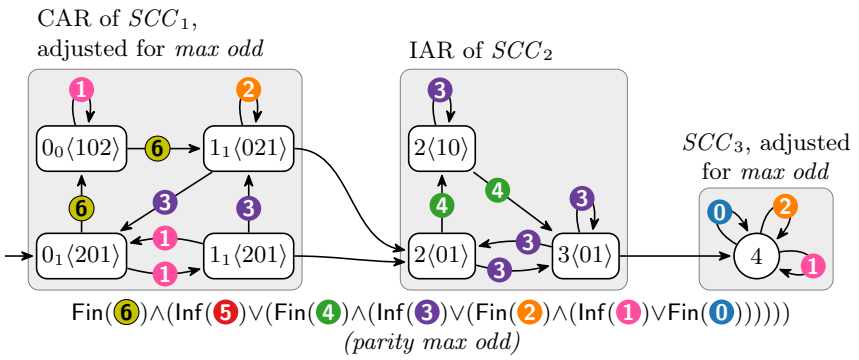


Fig. 7. Paritization of the automaton of Figure 5, combining the transformed SCCs of Figure 6 after adjustment to a common acceptance condition.

the order $\langle 102 \rangle$ is chosen as destination history for transition  : 1 is common to all edges going to 0_0, so we want it at the front of the history.

SCC-aware algorithms: These algorithms benefit from considering the SCCs of the original automaton. For CAR and IAR, the histories attached can be restricted to the colors present in the SCC [16]. The partial degeneralization needs not modify SCCs that do not contain all the colors C to degeneralize.

Heuristic ordering of colors to degeneralize: Our implementation of the partial degeneralization tries to guess, for each SCC, an appropriate ordering of the color to degeneralize: this is done by maintaining the order as a list of equivalence classes of colors, and refining this order as new transitions are processed. For instance if we degeneralize for the colors $C = \{0, 1, 2, 3\}$, the initial order will be $\langle\{0, 1, 2, 3\}\rangle$, then if the first transition we visit has colors $\{1, 3\}$ the new order will be refined to $\langle\{1, 3\}, \{0, 2\}\rangle$ and we jump to level 2 as we have now seen the first equivalence class of size 2.

Propagation of colors: To favor the grouping of colors in the dynamic ordering of the partial degeneralization, and in the history reuse optimization of IAR and CAR, we propagate colors as much as possible in SCCs. Ignoring transitions that are self-loops or that do not have both extremities in the same SCC, colors common to all incoming transitions of a state can be copied to all outgoing transitions and vice-versa. E.g.,  is seen as the equivalent  , showing that cycles with 1 always have 0.

The next section goes one step further in SCC-awareness, by actually simplifying the acceptance condition for each SCC according to the colors present. The paritization strategy to apply (CAR, IAR, identity, ...) can then be chosen independently for each SCC.

5 Paritization with Multiple Strategies

We now describe our paritization algorithm taking as input a TELA $\mathcal{A}$:

1. Enumerate the SCCs S_i of $\mathcal{A}$. For each S_i, perform the following operations:
   (a) Consider the sub-automaton $\mathcal{A}_{|S_i}$.
   (b) If $\mathscr{L}(\mathcal{A}_{|S_i})$ is empty [4], strip all colors and set the acceptance condition to $\bot$, which is a corner case for *parity max even* formula. (For *parity max* acceptances, transitions without color can be interpreted as having color -1.). Go back to the step 1a to process the next SCC or to step 2 if it does not exist.
   (c) Apply the simplifications described in section 3.
   (d) Transform the automaton $\mathcal{A}_{|S_i}$ into a *parity max* automaton R_i using the first applicable procedure from the following list:
      – Do nothing if the acceptance is already a *parity max* formula;
      – If the acceptance has the shape $\mathsf{Inf}(m_0) \vee (\mathsf{Fin}(m_1) \wedge (\mathsf{Inf}(m_2) \vee ...))$ of a *parity max*, renumber the colors $m_0, m_1, \ldots$ in decreasing order to get a *parity max* formula;

  - Adjust the condition to $\mathsf{Inf}(0)$ and the labeling of the transitions if this is a deterministic Rabin-like automaton that is Büchi-type (this requires a transition-based adaptation of an algorithm by Krishnan et al. [15]); note that $\mathsf{Inf}(0)$ is also a *parity max even* formula.
  - Dually, adjust the condition to $\mathsf{Fin}(0)$ if this is a deterministic Streett-like automaton that is co-Büchi-type, since $\mathsf{Fin}(0)$ is also a *parity max odd* formula.
  - Optionally, adjust the condition to parity and the labeling of the transitions if this is a deterministic automaton that is parity-type.

(e) If one of these steps has been applied, go back to the step 1a to process the next SCC or to step 2 if it does not exist.

(f) If the simplified acceptance condition contains conjunctions of Inf or disjunctions of Fin, apply the partial degeneralization construction (maybe multiple times) for all those terms, and remove unused colors. Since this incurs a blowup of the state-space that is linear (maybe multiple times) in the number of colors removed, it generally helps the CAR construction which has a worst case factorial blowup in the number of colors. Also, after this step, the acceptance condition might match more specialized algorithms in the next step. Jump to step 1c as the acceptance changed.

(g) Propagate colors in the SCC (Section 4.4).

(h) Transform the automaton $\mathcal{A}_{|S_i}$ into a *parity max* automaton R_i using the first applicable procedure from the following list:
  - If the automaton is Rabin-like or Streett-like, apply IAR to obtain a *parity max* automaton. When the acceptance formula can be interpreted as both Rabin-like or Streett-like we use the interpretation with the fewest number of pairs (cf. Remark 1).
  - Otherwise, apply CAR to obtain a *parity max* automaton.

2. Now that each automaton $\mathcal{A}_{|S_i}$ has been converted into an automaton R_i whose *parity* acceptance is either *max odd* or *max even*, adjust those acceptance conditions by incrementing or decrementing the colors of some R_i so that they can all use the same acceptance, and stitch all R_i together to form the final automaton R. For any transition of $\mathcal{A}$ that goes from state q in SCC i to state q' in SCC j, R should have a transition for each copy of q in R_i and going to one copy of q' in R_j. Similarly, the initial state of R should be any copy of the initial state of $\mathcal{A}$.
3. As a final cleanup, the number of colors of R can be reduced by computing the Rabin-index of the automaton [6].

Figures 5–7 show this algorithm at work on a small example with three SCCs. Figure 7 shows the result of step 2. Executing step 3 would reduce the number of colors to 2 (or to 3 if uncolored transitions are disallowed).

We now comment the details of Figure 6. The notation S+P refers to the Simplification of the acceptance condition (step 1c) and the Propagation of colors in the SCC (step 1g). On SCC_1, step 1c replaces 4 by 2, because these always occur together, and step 1g adds 2 on the transition from 1 to 0. After partial degeneralization, the sub-formula $\mathsf{Fin}(0) \wedge \mathsf{Fin}(4)$ can be fused into a single

$\mathsf{Fin}(0)$ (see Remark 2) by simply replacing 4 by 0 in the automaton, and after that the marks on the transitions before and after state 0_0 are propagated by step 1g. The resulting automaton is neither Rabin-like nor Streett-like, so it is transformed to parity using CAR; however the history of the states only have 3 colors to track instead of the original 5. In SCC_2, $\mathsf{Fin}(3)$ and $\mathsf{Inf}(4)$ can be replaced respectively by $\top$ and $\bot$ because 3 and 4 are not used. The acceptance condition is therefore reduced to the Rabin acceptance condition displayed, and IAR can be used instead of CAR. (Using CAR would build at least 4 states.) Finally SCC_3's acceptance conditions reduces to $\mathsf{Inf}(2) \wedge \mathsf{Fin}(1)$. Renumbering the colors to $\mathsf{Fin}(1) \wedge \mathsf{Inf}(0)$ gives us a parity max odd acceptance.

To stitch all these results together, as in Figure 7, we adjust all automata to use *parity max odd*: in SCC_1 this can be done for instance by decrementing all colorsand in SCC_3 by incrementing them (handling any missing color as -1).

Our implementation uses an additional optimization that we call the *parity prefix detection*. If the acceptance formula has the shape $\mathsf{Inf}(m_0) \vee (\mathsf{Fin}(m_1) \wedge (\mathsf{Inf}(m_2) \vee (...\beta)))$, i.e., it starts like a *parity max* formula but does not have the right shape because of β, we can apply CAR or IAR using only β while preserving the color $m_0, m_1, m_2, \ldots$ of the parity prefix, and later renumber all colors so the formula becomes *parity max*. This limits the colors that CAR and IAR have to track, so it reduces the number of states in the worst case.

Remark 4. The automaton used in the example 3 is the result of the propagation of colors on the example 2. We showed that the propagation can help the simplification and in practice, we apply a propagation before a simplification but in order to simplify our algorithm, we suppose that the propagation is not used to simplify the condition.

6 Experimental Evaluation

The simple CAR described in Section 4.1, without the optimizations of Section 4.4 was implemented in Spot 2.8 [8] as a function `to_parity()`. It can be used by Spot's `ltlsynt` tool with option `--algo=lar`; in that case the LTL specification φ passed to `ltlsynt` is converted to a deterministic TELA $\mathcal{A}_\varphi$ with arbitrary acceptance and then transformed into a parity automaton $\mathcal{P}_\varphi$ with `to_parity()` before the rest of the LTL synthesis procedure is performed.

The TELA $\mathcal{A}_\varphi$ built internally by `ltlsynt` can be obtained using Spot's `ltl2tgba -G -D` command: the construction is similar to the `delag` tool [22] and regards the original formula as a Boolean combination of LTL sub-formulas φ_i, translating each φ_i to a deterministic TELA $\mathcal{A}_{\varphi_i}$ (by combining classical LTL-to-generalized-Büchi translation [7] with specialized constructions for subclasses of LTL [10], or a Safra-based procedure [24]), and combining those results using synchronized products to obtain a TELA whose acceptance condition is the Boolean combination of the acceptance conditions of all the $\mathcal{A}_{\varphi_i}$.

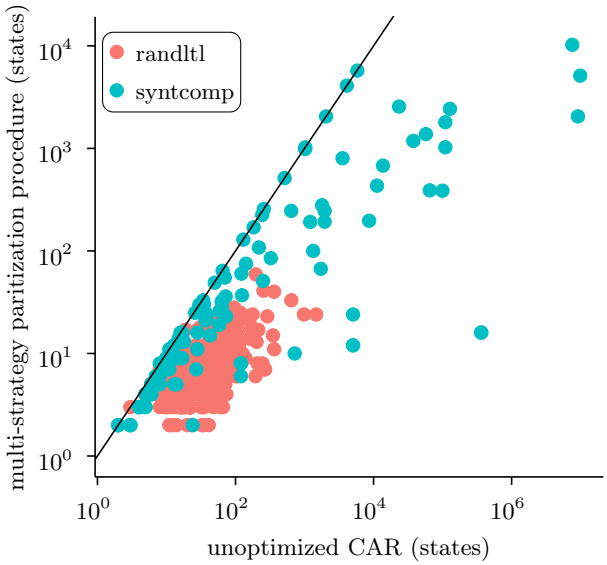


Fig. 8. Comparison of the new multi-strategy paritization (Section 5) against the unoptimized CAR (Section 4.1)

In Spot 2.9, `to_parity()` was changed to implement Section 5 and the optimizations of Section 4.4. We are therefore in position to compare the improvements brought by those changes on the transformation of $\mathcal{A}_\varphi$ to $\mathcal{P}_\varphi$. [1]

We evaluate the improvements on two sets of automata:

syntcomp contains automata generated with `ltl2tgba -G -D` from LTL formulas from the sequential TLSF track of SyntComp'2020. Among those automata, we have removed those that already had a parity acceptance (usually Büchi acceptance). The remaining set contains 25 automata with a generalized-Büchi condition, 1 with a co-Büchi condition, 3 with a generalized Rabin condition, 17 with a generalized Streett condition and 84 with a condition that mixes Fin and Inf terms. The average number of accepting SCCs is 100.2 (min. 1, med. 2, max. 5741). The average number of states is 1966.1 (min. 1, med. 16.5, max. 81921).

randltl contains 346 automata built similarly, from random LTL formulas. Furthermore, we have ensured that no automaton has parity acceptance, and all of them use at least 5 colors (med. 5, avg. 5.2, max. 10). The average number of accepting SCCs is 1.7 (min. 1, med. 1, max. 16). The average number of states is 5.6 (min. 1, med. 4, max. 41). Only 21 of these automata have a Rabin-like or Streett-like acceptance condition, 9 have a generalized Rabin condition, 1 has a generalized Streett condition.

The improvement of our new paritization based on multiple strategies over our old unoptimized CAR implementation is shown on Figure 8.

[1] To reproduce these results, see https://www.lrde.epita.fr/~frenkin/atva20/

6.1 Influence of options

Table 6.1 selectively disables some optimizations to evaluate their effect on the number of output states. Configuration "all − x" means that optimization x is disabled. *Rabin to Büchi* is the detection of Rabin-like (or Streett-like) automata that are Büchi (or co-Büchi) realizable at step 1d. *Parity prefix* is the optimization mentioned at the very end of Section 5. *Simplify acc*, *propagate colors*, and *partial degen* correspond respectively to steps 1c and 1g, and 1f. We can see that enabling only partial degeneralization ("nothing + partial degen") is better than enabling all other options except partial- degeneralization ("all − partial degen"). No other optimization has such a large effect.

The scatter plot of Figure 8 compares the cases summarized by lines *all* and *unoptimized CAR*, while Figure 9 plots the data of lines *all* and *all − partial degen*.

Partial degeneralization appears to be the most important optimization, because in addition to reducing the number of colors, it may help to use IAR or even simpler construction. The propagation of colors, which allows more flexibility in the selection of histories, is the second best optimization. *Hist. reuse* corresponds to the history reuse described in Section 4.4. *all − reuse latest* has history reuse enabled, but uses the oldest compatible state instead of the latest — hence our heuristic of using the latest compatible state. Finally *Unoptimized CAR* is a straightforward implementation of CAR given for comparison.

When partial degeneralization is not Desirable Applying the partial degeneralization with m colors may remove $m-1$ colors, and multiply the number of states by m in the worst case. Applying CAR on an automaton with n colors will multiply the number of states by $n!$. Thus, in order to handle the worst case scenario, partial degeneralization should be applied before CAR whenever possible. For instance if we did not perform the partial degeneration on SCC_1 of Figure 6 (page 20), CAR would have tracked four colors and would have built a 6-state automaton.

Another argument in favor of doing a degeneralization is that it may help shape the acceptance condition into something that is easier to paritize. As an example, it may allow us to use IAR instead of CAR. From this perspective, it can be useful to use a partial degeneralization even if it does not reduce the number of colors of the automaton. However if the partial degeneralization process failed to reduce the number of colors, and we still have to use CAR, then it is better to apply CAR on the smaller, non-partially-degeneralized automaton.

A significant difference between the CAR/IAR procedures and the partial degeneralization is that the latter has to fix an ordering of the colors. This way it can keep track of the colors encountered using only a counter (an index in the order) instead of a subset of colors, but this works best if the colors occur in this order. Even if our implementation uses heuristics to select a more suitable ordering for partial degeneralization (cf. Section 4.4), it may be the case that it fails to find a good ordering, or that there is no good order for the entire automaton.

Table 2. Effect of various optimizations on the paritization procedure. Configuration "all" corresponds to the algorithm of section 5. Configuration "unoptimized CAR" corresponds to the basic CAR implementation of section 4.1. Configuration "nothing" implements only the SCC-based paritization.

dataset	randltl		syntcomp		both	
config	amean	gmean	amean	gmean	amean	gmean
default + parity type	7.647	6.302	246.2	27.15	64.39	8.920
default + Büchi type	7.919	6.525	246.2	27.15	64.59	9.159
default	7.965	6.560	246.2	27.15	64.63	9.197
default - Rabin to Büchi	7.965	6.560	246.2	27.15	64.63	9.197
default - custom order	7.928	6.563	246.2	27.15	64.60	9.200
default - parity equiv. - parity prefix	7.986	6.580	246.2	27.15	64.64	9.218
default - parity prefix	7.986	6.580	246.2	27.15	64.64	9.218
default + TAR	8.130	6.652	246.2	27.15	64.75	9.295
default - hist. reuse	8.364	6.828	246.2	27.15	64.93	9.481
default - IAR	8.156	6.767	254.3	28.92	66.70	9.560
default - simplify acc.	8.873	6.946	246.2	27.15	65.32	9.606
default - BSCC	8.413	6.954	246.2	27.16	64.98	9.616
default - reuse last	8.653	7.089	246.2	27.15	65.15	9.757
default - partial degen	8.017	6.496	3264.3	50.16	782.65	10.564
default - propagate colors	22.584	11.982	246.2	27.15	75.77	14.555
nothing + parity type	28.613	16.323	6201.3	71.59	1497.00	23.203
nothing + propagate colors	22.098	16.262	5705.6	92.09	1374.11	24.565
nothing + partial degen	47.049	27.746	319.1	37.11	111.77	29.733
nothing + BSCC	47.179	25.599	6181.6	80.31	1506.47	33.600
nothing + Büchi type	51.081	27.975	6205.6	77.77	1515.16	35.678
nothing + simplify acc.	51.962	29.461	6183.2	76.92	1510.49	37.016
nothing + custom order	52.283	29.798	6046.3	92.98	1478.17	39.061
unoptimized CAR	55.260	31.280	6357.4	82.82	1554.46	39.433
nothing + hist. reuse (last)	54.777	31.613	5814.8	91.34	1425.01	40.690
nothing + parity prefix	54.121	31.609	6206.2	96.04	1517.61	41.173
nothing + IAR	56.523	33.109	7120.6	95.34	1736.96	42.581
nothing + hist. reuse (first)	58.309	34.420	6194.9	94.93	1518.13	43.815
nothing + Rabin to Büchi	58.931	35.130	6329.1	100.19	1550.52	45.076
nothing + parity equiv.	59.552	35.534	6582.5	103.33	1611.26	45.806
nothing	59.552	35.534	6582.5	103.33	1611.26	45.806
nothing + TAR	63.191	41.032	6591.4	103.51	1616.17	51.134

dataset	randltl		syntcomp		both	
config	amean	gmean	amean	gmean	amean	gmean
default - parity - prefix	0.09074	0.09062	3.214	0.1845	0.9439	0.1100
default - propagate colors	0.09104	0.09092	3.108	0.1829	0.9151	0.1101
nothing + partial degen	0.09023	0.09009	3.791	0.1890	1.1010	0.1103
default - Rabin to Büchi	0.09068	0.09058	3.213	0.1870	0.9434	0.1104
default - hist. reuse	0.09162	0.09147	3.208	0.1829	0.9428	0.1105
default - parity equiv. - parity prefix	0.09264	0.09220	4.247	0.1929	1.2273	0.1128
default - reuse last	0.10009	0.09751	3.225	0.1909	0.9535	0.1171
default - BSCC	0.09826	0.09769	3.256	0.1934	0.9608	0.1177
default - simplify acc.	0.09805	0.09776	3.192	0.1935	0.9430	0.1178
default - custom order	0.09896	0.09864	3.255	0.1961	0.9608	0.1190
default - IAR	0.09962	0.09905	3.260	0.1965	0.9626	0.1194
default	0.09855	0.09824	3.287	0.2070	0.9695	0.1204
default + TAR	0.10857	0.10190	3.217	0.1913	0.9574	0.1210
nothing + hist. reuse (last)	0.09093	0.09079	8.440	0.3578	2.3711	0.1320
nothing + parity prefix	0.09169	0.09151	10.735	0.3656	2.9986	0.1336
nothing + simplify acc.	0.09039	0.09022	10.563	0.3807	2.9506	0.1337
nothing + propagate colors	0.09185	0.09170	10.693	0.3651	2.9871	0.1337
nothing + TAR	0.09200	0.09181	10.700	0.3701	2.9890	0.1343
nothing + hist. reuse (first)	0.09138	0.09123	8.439	0.3780	2.3712	0.1345
nothing	0.09152	0.09138	10.701	0.3778	2.9890	0.1346
nothing + IAR	0.09069	0.09055	10.900	0.3861	3.0427	0.1346
nothing + Rabin toBüchi	0.09208	0.09192	10.711	0.3789	2.9921	0.1353
nothing + parity equiv.	0.09396	0.09369	10.698	0.3681	2.9901	0.1361
nothing + BSCC	0.09006	0.08989	10.722	0.4282	2.9937	0.1377
nothing + custom order	0.09144	0.09128	54.769	0.4636	15.0244	0.1423
default - partial degen	0.09185	0.09165	82.190	0.5098	22.5135	0.1464
default + Büchi type	0.10220	0.09792	109.179	0.5023	29.8920	0.1530
default + parity type	0.11004	0.10351	113.414	0.4730	31.0545	0.1568
nothing + parity type	0.09118	0.09099	125.426	0.7183	34.3213	0.1600
nothing + Büchi type	0.09203	0.09180	121.184	0.7027	33.1633	0.1601
unoptimized CAR	18.27499	18.27497	1117.727	51.6140	318.5455	24.2664

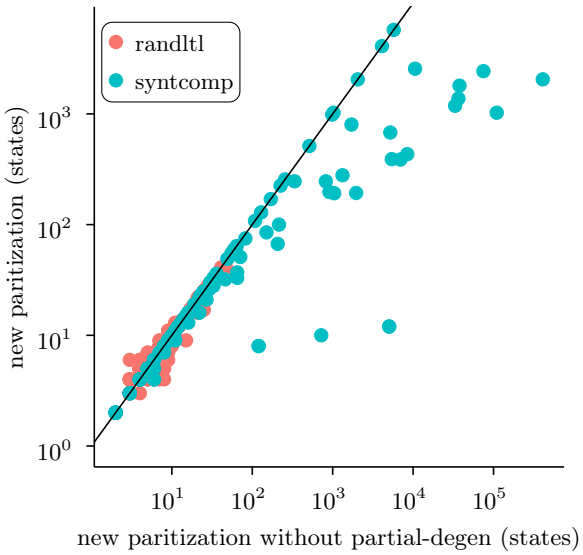
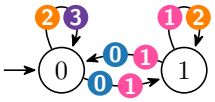


Fig. 9. Effect of disabling the partial degeneralization in the new paritization.

Comparing the number of states produced by our paritization procedure with and without partial-generalization reveals a few cases where partial degeneralization is actually harmful. See the few dots above the diagonal in Figure 9.

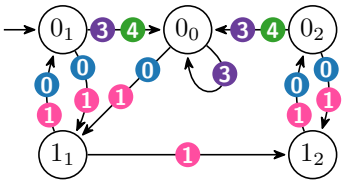
The following automaton illustrates a case where partial degeneralization produces an automaton larger than direct application of CAR.

Consider the following input automaton:



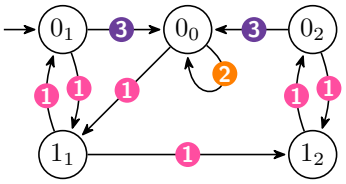
$\big(\mathsf{Fin}(3) \wedge (\mathsf{Inf}(1) \vee \mathsf{Fin}(0))\big) \vee \big(\mathsf{Inf}(0) \wedge \mathsf{Inf}(2) \wedge \mathsf{Inf}(3)\big)$

Applying partial degeneralization on $\{0, 2, 3\}$ yields the following automaton. (The ordering of colors chosen heuristically is $d_0 = 0$, $d_1 = 2$, and $d_2 = 3$.)



$\big(\mathsf{Fin}(3) \wedge (\mathsf{Inf}(1) \vee \mathsf{Fin}(0))\big) \vee \mathsf{Inf}(4)$

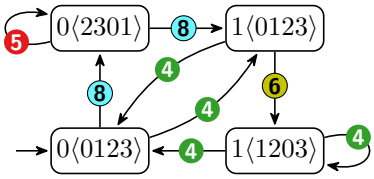
By chance, the shape of the acceptance formula corresponds to a *parity max odd* formula, up to a renaming of colors. After renaming 3, 4 to 2, 3, and keeping only the maximum color on each edge, we end up with a parity automaton:



$\mathsf{Inf}(3) \vee \big(\mathsf{Fin}(2) \wedge (\mathsf{Inf}(1) \vee \mathsf{Fin}(0))\big)$
parity max odd

The number of colors can be further lowered, but the point here is that using the partial degeneralization created a 5 state parity automaton, and "saved" us from using CAR or IAR.

However, in this case, had was used CAR directly on the input automaton, we would have produced the following 4-state parity automaton instead:



$\mathsf{Inf}(8) \vee (\mathsf{Fin}(7) \wedge (\mathsf{Inf}(6) \vee (\mathsf{Fin}(5) \wedge (\mathsf{Inf}(4) \vee (\mathsf{Fin}(3) \wedge (\mathsf{Inf}(2) \vee (\mathsf{Fin}(1) \wedge \mathsf{Inf}(0))))))))$
parity max odd

While this example shows that CAR's performance may sometimes improve if we don't run the partial degeneralization first, our experiments suggest that it is more often better to use it.

Note that this specific automaton was fixed by implementing[2] an additional simplification of the acceptance condition. For instance, in the input automaton, one can notice that 3 always appears along with 2, thus the formula $\mathsf{Inf}(2) \wedge \mathsf{Inf}(3)$ can be reduced to $\mathsf{Inf}(3)$. Both CAR and partial degeneralization would then produce a 3-state parity automaton.

Conversion to a generalized-Rabin condition As described in section 4.2, we prefer to apply IAR instead of CAR when possible. In order to do it, just before the step 1f of the algorithm, we convert $\mathcal{A}_{S_i}$ to a generalized-Rabin automaton. Then the partial degeneralization gives us a Rabin automaton that can be converted to a parity automaton by using IAR.

In the table 6.1, we describe on the same set of automata than before the influence of this option on the time needed to paritize the automata. We see that there is no influence for the set of automata created from a set of random LTL formula but on the set of automaton converted from the LTL formula of the Syntcomp, disabling this option increases the geometric mean by 10 percent while the arithmetic mean increases by 4 percent. Now if we take the table 6.1, the results are more contrasted. We see that using a generalized-Rabin automaton reduces by 6 percent the geometric mean of the number of states on the set randltl while on the set syntcomp it decreases by 7 percent.

Why parity-type and Büchi-type are not enabled? In the table 6.1, we see that our automata are smaller when we try to convert a parity-type automaton to

[2] https://gitlab.lrde.epita.fr/spot/spot/-/issues/406

Table 3. Effect of the conversion to a generalized-Rabin automaton before using LAR on the time of paritization.

dataset	randltl		syntcomp		both	
config	amean	gmean	amean	gmean	amean	gmean
With generalized-Rabin	0.11	0.10	3.14	0.19	0.94	0.12
Without generalized-Rabin	0.10	0.10	3.29	0.21	0.97	0.12

Table 4. Effect of the conversion to a generalized-Rabin automaton before using LAR on the number of states of the resulting DPA.

dataset	randltl		syntcomp		both	
config	amean	gmean	amean	gmean	amean	gmean
With generalized-Rabin	7.66	6.16	12122.56	66.88	3316.36	11.82
Without generalized-Rabin	7.97	6.56	12116.84	62.48	3315.01	12.14

a parity automaton than without this option. This option is disabled by default because the complexity is too great to be applied on large automata in practice.

We start by describing how we convert a Büchi-type automaton to a Büchi-automaton. The principle is to search all the transitions that are always accepting. In Spot, we had an implementation for Rabin as it is easy to find such transitions. In order to apply it to the general case, we adapted an emptiness check algorithm to get all the transitions that are in at least one accepting cycle. The idea is that we take the dual of the condition of the original automaton, any transition that is not in any accepting cycle of this automaton will be always accepting in the original automaton. If we assign to this set of transitions the color 0 and we use a Büchi condition, we get an automaton that recognizes a subset of the language of the original automaton and we just have to test the inclusion of the original automaton to determine if the result is equivalent to the original. In this procedure, the hardest part is to find which transitions are accepting.

When we convert a parity-type automaton (we suppose parity max even), we use the same idea. All the transitions that are always accepting get the maximal even colors. Those transitions are removed and we search the always rejecting transitions in the resulting automaton. Those transitions get an odd color lower than the previous one. We repeat this process while we have uncolored transitions. The resulting automaton is equivalent if is there is not any uncolored transition.

In the figure 6.1, we see that for many automata that the algorithm can process in more than one second, trying to convert a parity type automaton leads to the algorithm not finishing in less than 1000s.

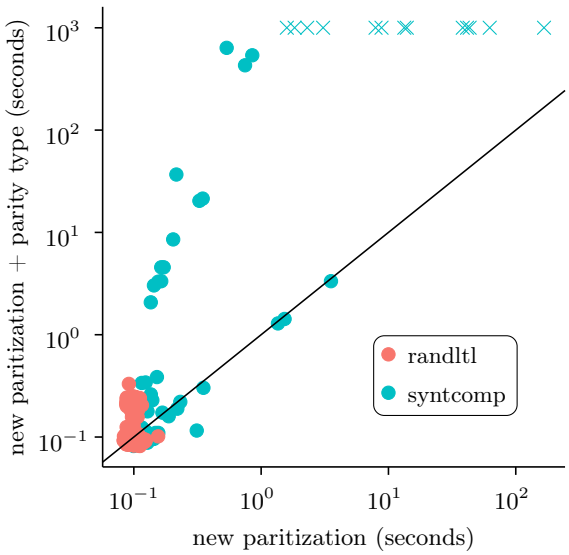


Fig. 10. Comparison of the multi-strategy paritisation against the version that tries to convert parity-type automata. Crosses are the cases that were not done in less than 1000s.

6.2 Application to reactive synthesis

To assert the effect of the improved paritization on `ltlsynt`, we ran the entire SyntComp'20 benchmark (including formulas omitted before) with a timeout of `1000 seconds`, and counted the number of cases solved by different configurations of `ltlsynt`, as reported in Table 5. We can see that improving CAR with all the tricks of Section 5 allowed the `ltlsynt`'s LAR-based approach to perform better than `ltlsynt`'s Safra-based approaches.

While table 5 shows that the new version of LAR solves more cases than the other configurations of `ltlsynt`, it does not actually tell whether the set of cases solved is a superset of the other configurations or not. Table 7 reveals that despite the number of solved cases increasing, some cases are solved by other configurations but not by the new LAR.

7 Conclusion

We have presented a procedure that converts any TELA into a transition-based parity automaton. Our algorithm combines algorithms that are transition-based adaptations or generalizations of known procedures (e.g., CAR is a adaption of the classical SAR and partial degeneration extends the standard generalization technique), thus this paper can also be read as a partial survey of acceptance condition transformations presented under a unified framework.

Table 5. Number of SyntComp'20 cases solved by `ltlsynt` under different configurations, with a timeout of 1000 seconds. PAR-2 (penalized average runtime) sums the time of all successful instances, plus twice the timeout for unsuccessful ones.

option	approach to paritization	# solved	PAR-2
`--algo=sd`	LTL to Büchi, then split input/output variables, then Safra-based determinization [21]	255	206014s
`--algo=ps`	LTL to DPA, then split input/output variables	282	152451s
`--algo=lar.old`	LTL to determ. TELA, then CAR of Section 4.1	287	142032s
`--algo=ds`	LTL to Büchi, then Safra-based determinization, then split input/output variables [21]	292	132308s
`--algo=lar`	LTL to determ. TELA, then approach of Section 5	309	98505s

Table 6. Number of cases of the SyntComp'20 for which at least one configuration of `ltlsynt` with the option `synthesis` succeeded and at least one configuration failed. The number n at line i and column j means that the configuration i solved n cases that j was not able to do.

	`lar`	`lar.old`	`ds`	`sd`	`ps`
`lar`	-	22	21	58	27
`lar.old`	0	-	5	41	7
`ds`	4	10	-	39	10
`sd`	4	9	2	-	7
`ps`	0	2	0	34	-

The CAR construction, which is the general case for our paritization algorithm, produces smaller automata than the classical SAR, as it tracks colors instead of states, and uses transition-based acceptance. We further improved this construction by applying more specialized algorithms in each SCC (IAR [16], detection of Büchi-realizable SCCs [15], detection of empty SCCs [4], detection of parity) after simplifying their acceptance.

The proposed partial degeneralization procedure is used as a preprocessing step to reduce conjunctions of Inf or disjunction of Fin in the acceptance condition, and to reduce the number of colors that CAR and IAR have to track. Since partial degeneralization only causes a linear blowup in the number of colors removed, it generally helps the CAR construction whose worst case scenario incurs a factorial blowup in the number of colors. Furthermore, after partial degeneralization, the acceptance condition may match more specialized algorithms.

The implementation of the described paritization procedure is publicly available in Spot 2.9. While our motivation stems from one approach to produce deterministic parity automata used in Spot, this paritization also works with non-deterministic automata: it preserves the determinism of the input.

Acknowledgment. The unoptimized CAR definition of Section 4.1 was first implemented in Spot by Maximilien Colange.

References

1. S. Baarir and A. Duret-Lutz. SAT-based minimization of deterministic ω-automata. In *LPAR'15*, *LNCS* 9450, pp. 79–87. Springer, 2015.
2. T. Babiak, T. Badie, A. Duret-Lutz, M. Křetínský, and J. Strejček. Compositional approach to suspension and other improvements to LTL translation. In *SPIN'13*, *LNCS* 7976, pp. 81–98. Springer, 2013.
3. 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*, *LNCS* 8172, pp. 442–445. Springer, 2015. See also `http://adl.github.io/hoaf/`.
4. C. Baier, F. Blahoudek, A. Duret-Lutz, J. Klein, D. Müller, and J. Strejček. Generic emptiness check for fun and profit. In *ATVA'19*, *LNCS* 11781, pp. 445–461. Springer, 2019.
5. R. Bloem, K. Chatterjee, and B. Jobstmann. *Graph Games and Reactive Synthesis*, chapter 27, pp. 921–962. Springer, 2018.
6. O. Carton and R. Maceiras. Computing the Rabin index of a parity automaton. *Informatique théorique et applications*, 33(6):495–505, 1999.
7. A. Duret-Lutz. LTL translation improvements in Spot 1.0. *International Journal on Critical Computer-Based Systems*, 5(1/2):31–54, 2014.
8. A. Duret-Lutz, A. Lewkowicz, A. Fauchille, T. Michaud, E. Renault, and L. Xu. Spot 2.0 — a framework for LTL and ω-automata manipulation. In *ATVA'16*, *LNCS* 9938, pp. 122–129. Springer, 2016.
9. E. A. Emerson and C.-L. Lei. Modalities for model checking: Branching time logic strikes back. *Science of Computer Programming*, 8(3):275–306, 1987.
10. J. Esparza, J. Křetínský, and S. Sickert. One theorem to rule them all: A unified translation of LTL into ω-automata. In *LICS'18*, pp. 384–393. ACM, 2018.
11. B. Farwer. *ω-Automata*, *LNCS* 2500, chapter 1, pp. 3–20. Springer, 2001.
12. P. Gastin and D. Oddoux. Fast LTL to Büchi automata translation. In *CAV'01*, *LNCS* 2102, pp. 53–65. Springer, 2001.
13. J. Křetínský and J. Esparza. Deterministic automata for the (F,G)-fragment of LTL. In *CAV'12*, *LNCS* 7358, pp. 7–22. Springer, 2012.
14. J. Kretínský, T. Meggendorfer, and S. Sickert. Owl: A library for ω-words, automata, and LTL. In *ATVA'18*, *LNCS* 11138, pp. 543–550. Springer, 2018.
15. S. C. Krishnan, A. Puri, and R. K. Brayton. Deterministic ω-automata vis-a-vis deterministic Büchi automata. In *ISAAC'94*, *LNCS* 834, pp. 378–386. Springer, 1994.
16. J. Křetínský, T. Meggendorfer, C. Waldmann, and M. Weininger. Index appearance record for transforming Rabin automata into parity automata. In *TACAS'17*, *LNCS* 10205, pp. 443–460, 2017.
17. C. Löding. Methods for the transformation of ω-automata: Complexity and connection to second order logic. Diploma thesis, Institue of Computer Science and Applied Mathematics, 1998.
18. C. Löding. Optimal bounds for transformations of omega-automata. In *FSTTCS'99*, *LNCS* 1738, pp. 97–109. Springer, 1999.
19. M. Luttenberger, P. J. Meyer, and S. Sickert. Practical synthesis of reactive systems from LTL specifications via parity games. *Acta Informatica*, 57:3—-36, 2020. Originally published on 21 November 2019.
20. J. Major, F. Blahoudek, J. Strejcek, M. Sasaráková, and T. Zboncáková. `ltl3tela`: LTL to small deterministic or nondeterministic Emerson-Lei automata. In *ATVA'19*, *LNCS* 11781, pp. 357–365. Springer, 2019.

21. T. Michaud and M. Colange. Reactive synthesis from LTL specification with Spot. In *SYNT'18*, 2018. URL http://www.lrde.epita.fr/dload/papers/michaud.18.synt.pdf.
22. D. Müller and S. Sickert. LTL to deterministic Emerson-Lei automata. In *GandALF'17*, vol. 256 of *EPTCS*, pp. 180–194, 2017.
23. N. Piterman. From nondeterministic büchi and streett automata to deterministic parity automata. *Logical Methods in Computer Science*, 3(3), 2007.
24. R. Redziejowski. An improved construction of deterministic omega-automaton using derivatives. *Fundamenta Informaticae*, 119(3-4):393–406, 2012.
25. S. Safra and M. Y. Vardi. On ω-automata and temporal logic. In *STOC'89*, pp. 127–137. ACM, 1989.

A Differences Between `ltlsynt` Configurations

Table 7: Specifications of the SyntComp'20 benchmark for which at least one configuration of `ltlsynt` with the option `synthesis` succeeded (✓) and at least one configuration failed (−).

file	sd	ps	lar.old	ds	lar
ltl2dba_beta_10.tlsf	✓	−	−	−	−
ModdifiedLedMatrix4X.tlsf	✓	−	−	−	−
amba_case_study_2.tlsf	−	−	−	✓	−
collector_v3_7.tlsf	−	−	−	✓	−
ltl2dba_beta_8.tlsf	✓	−	−	✓	−
tictactoe.tlsf	✓	−	−	✓	−
detector_10.tlsf	−	−	−	−	✓
detector_12.tlsf	−	−	−	−	✓
detector_8.tlsf	−	−	−	−	✓
full_arbiter_enc_10.tlsf	−	−	−	−	✓
full_arbiter_enc_12.tlsf	−	−	−	−	✓
full_arbiter_enc_8.tlsf	−	−	−	−	✓
ltl2dba_C2_10.tlsf	−	−	−	−	✓
ltl2dba_C2_12.tlsf	−	−	−	−	✓
ltl2dba_C2_8.tlsf	−	−	−	−	✓
ltl2dba_theta_10.tlsf	−	−	−	−	✓
ltl2dba_theta_12.tlsf	−	−	−	−	✓
ltl2dba_theta_8.tlsf	−	−	−	−	✓
prioritized_arbiter_10.tlsf	−	−	−	−	✓
prioritized_arbiter_enc_10.tlsf	−	−	−	−	✓
prioritized_arbiter_enc_12.tlsf	−	−	−	−	✓
prioritized_arbiter_enc_8.tlsf	−	−	−	−	✓
lilydemo18.tlsf	−	−	✓	−	✓
ltl2dba_Q_10.tlsf	−	−	✓	−	✓
ltl2dba_theta_6.tlsf	−	−	✓	−	✓

ltl2dba_U1_10.tlsf	–	–	✓	–	✓
ltl2dba_U1_12.tlsf	–	–	✓	–	✓
simple_arbiter_enc_12.tlsf	–	–	–	✓	✓
prioritized_arbiter_8.tlsf	✓	–	–	✓	✓
simple_arbiter_enc_10.tlsf	✓	–	–	✓	✓
simple_arbiter_enc_8.tlsf	✓	–	–	✓	✓
simple_arbiter_10.tlsf	✓	✓	–	✓	✓
simple_arbiter_12.tlsf	✓	✓	–	✓	✓
amba_decomposed_arbiter_6.tlsf	–	–	✓	✓	✓
round_robin_arbiter_6.tlsf	–	–	✓	✓	✓
amba_decomposed_arbiter_2.tlsf	–	✓	✓	✓	✓
amba_decomposed_arbiter_4.tlsf	–	✓	✓	✓	✓
amba_decomposed_tburst4.tlsf	–	✓	✓	✓	✓
collector_v3_2.tlsf	–	✓	✓	✓	✓
collector_v3_3.tlsf	–	✓	✓	✓	✓
collector_v3_4.tlsf	–	✓	✓	✓	✓
collector_v3_5.tlsf	–	✓	✓	✓	✓
collector_v3_6.tlsf	–	✓	✓	✓	✓
lilydemo20.tlsf	–	✓	✓	✓	✓
lilydemo22.tlsf	–	✓	✓	✓	✓
lilydemo24.tlsf	–	✓	✓	✓	✓
load_balancer_10.tlsf	–	✓	✓	✓	✓
load_balancer_4.tlsf	–	✓	✓	✓	✓
load_balancer_6.tlsf	–	✓	✓	✓	✓
load_balancer_8.tlsf	–	✓	✓	✓	✓
loadcomp2.tlsf	–	✓	✓	✓	✓
loadcomp3.tlsf	–	✓	✓	✓	✓
loadcomp4.tlsf	–	✓	✓	✓	✓
loadcomp5.tlsf	–	✓	✓	✓	✓
loadfull2.tlsf	–	✓	✓	✓	✓
loadfull3.tlsf	–	✓	✓	✓	✓
loadfull4.tlsf	–	✓	✓	✓	✓
loadfull5.tlsf	–	✓	✓	✓	✓
ltl2dba_E_12.tlsf	–	✓	✓	✓	✓
round_robin_arbiter_2.tlsf	–	✓	✓	✓	✓
round_robin_arbiter_3.tlsf	–	✓	✓	✓	✓
round_robin_arbiter_4.tlsf	–	✓	✓	✓	✓
round_robin_arbiter_5.tlsf	–	✓	✓	✓	✓
round_robin_arbiter_unreal1_2_12.tlsf	–	✓	✓	✓	✓
round_robin_arbiter_unreal1_2_15.tlsf	–	✓	✓	✓	✓
round_robin_arbiter_unreal1_2_18.tlsf	–	✓	✓	✓	✓
round_robin_arbiter_unreal1_2_3.tlsf	–	✓	✓	✓	✓
round_robin_arbiter_unreal1_2_6.tlsf	–	✓	✓	✓	✓
round_robin_arbiter_unreal1_2_9.tlsf	–	✓	✓	✓	✓