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 1 and 3 on page 9 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 [4]. The high-level approach taken by our `ltlsynt` tool [20], or even by the SyntComp'19 winner Strix [18], 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 [22, 23]. This is the current approach of `ltlsynt` [20]. However, since the introduction of the HOA format [2] allowing the representation of TELA, we have seen the development of several tools for converting LTL formulas into TELA: for instance `delag` [21], `ltl2da` and `ltl2na` (all three part of newer versions of Owl [13]), `ltl3tela` [19], or Spot's `ltl2tgba -G` (see Section 5), 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*) [16, 17] 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*) [16, 17] 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 [10]). We describe the adaptation of these two procedures in Section 3. 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. [15]. A third transformation, also described in Section 3, can be used as a preprocessing before the previous procedures: this is a *partial degeneralization*, i.e. an extension of the classical degeneralization procedure [11, 1] 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 4 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 5 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 [8] and named [24] in the 80s, and 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* [2]. Instead of working directly with sets of transitions, we label transitions by multiple colored marks, as can be seen in Figures 1–3.

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 $N \models \alpha$ according to the following semantics:

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

Intuitively, an Emerson-Lei automaton is an ω-automaton labeled by marks and whose acceptance condition is expressed as a positive Boolean formula on sets of marks that occur infinitely often 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, and* $\alpha \in \mathcal{C}(M)$ *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 **rep**eated 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 $(\ell_i)_{i \geq 0} \in \Sigma^\omega$. 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.

Example 1. In the automaton of Figure 1, 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.

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.

The only unusual formulas of 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 $\bigvee_i \big(\mathsf{Fin}(m_{2i}) \wedge \mathsf{Inf}(m_{2i+1})\big) \vee \bigvee_j (\mathsf{Fin}(a) \wedge$

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	$\bigwedge_i (\mathsf{Inf}(m_{2i}) \vee \mathsf{Fin}(m_{2i+1})) \wedge \bigwedge_j \mathsf{Inf}(m_j) \wedge \bigwedge_k \mathsf{Fin}(m_k)$
Cf. Appendix A	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	$\mathsf{Inf}(2k+1) \vee (\mathsf{Fin}(2k) \wedge (\mathsf{Inf}(2k-1) \vee (\mathsf{Fin}(2k-2) \wedge \ldots)))$

$\mathsf{Inf}(m_j)) \vee \bigvee_k (\mathsf{Fin}(m_k) \wedge \mathsf{Inf}(b))$ 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).

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 $\mathscr{L}(\mathcal{A}_{|S}) \neq \emptyset$.*

3 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 3.1) works for any input, while IAR (Section 3.2) is specialized for

Rabin-like or Streett-like inputs. The third algorithm is a *partial degeneralization* (Section 3.3): it takes an automaton with any acceptance formula α, and produces an automaton where any generalized Büchi (resp. generalized co-Büchi) subformula of α have been replaced by a Büchi (resp. co-Büchi) formula. Optimizations common to these algorithms are listed in Section 3.4.

3.1 Color Appearance Record

Consider a set of marks $M = \{0, 1, \ldots, n-1\}$ and a TELA $\mathcal{A} = (Q, M, \Sigma, \delta, q_0, \alpha)$. Let $\Pi(M)$ be the set of *permutations* of M. We can represent a permutation $\sigma \in \Pi(M)$ by a table $\langle \sigma(0), \sigma(1), \ldots, \sigma(n-1) \rangle$.

The *Color Appearance Record* (CAR) algorithm pairs such permutations of colors with states of the input automaton in order to keep track of the history of colors visited in the corresponding run of $\mathcal{A}$, in the order they were last seen. Output states are therefore elements of $Q^{\mathrm{CAR}} = Q \times \Pi(M)$.

We *update* histories with a function $\mathcal{U} : \Pi(M) \times M \to \Pi(M) \times 2^M$, such that $\mathcal{U}(\sigma, c) = (\langle c, \sigma(0), \sigma(1), ..., \sigma(i-1), \sigma(i+1), ..., \sigma(n-1) \rangle, \{\sigma(0), \sigma(1), ..., \sigma(i)\})$ where $i = \sigma^{-1}(c)$ is the position of color c in σ. In other words, $\mathcal{U}(\sigma, c)$ moves c to the front of σ by rotating the first $i+1$ elements: it returns the new permutation and the set of rotated elements. This update function can be generalized to a set of colors recursively as follows:

$$\widetilde{\mathcal{U}}(\sigma, \emptyset) = (\sigma, \emptyset)$$
$$\widetilde{\mathcal{U}}(\sigma, \{c\} \cup C) = (\rho, R \cup S) \text{ where } (\pi, R) = \widetilde{\mathcal{U}}(\sigma, C) \text{ and } (\rho, S) = \mathcal{U}(\pi, c)$$

That is to say, $\widetilde{\mathcal{U}}(\sigma, C)$ moves the colors in C to the front of σ and also returns set of colors corresponding to the updated prefix. The order in which colors in C are moved to the front of σ is unspecified and may affect the size of the output automaton (see Section 3.4).

Let $M' = \{0, \ldots, 2n+1\}$ be the output marks. We define the transition relation $\delta^{\mathrm{CAR}} \subseteq Q^{\mathrm{CAR}} \times \Sigma \times 2^{M'} \times Q^{\mathrm{CAR}}$ as follows:

$$\delta^{\mathrm{CAR}} = \left\{ (q, \sigma) \xrightarrow{x, \{c\}} (q', \pi) \;\middle|\; q \xrightarrow{x, C} q' \in \delta,\ (\pi, R) = \widetilde{\mathcal{U}}(\sigma, C),\ c = 2|R| + [R \not\models \alpha] \right\}$$

where $[R \not\models \alpha]$ is a shorthand for 0 if $R \models \alpha$ and for 1 if $R \not\models \alpha$.

Theorem 1. *For any TELA $\mathcal{A} = (Q, M, \Sigma, \delta, q_0, \alpha)$ over the marks $M = \{0, \ldots, n-1\}$, there exists an equivalent TELA $\mathcal{A}' = (Q^{CAR}, M', \Sigma, \delta^{CAR}, (q_0, \pi_0), \alpha')$ where α' is a parity max even formula over $2n+1$ colors. The initial permutation can be any $\pi_0 \in \Pi(M)$.*

The proof is similar to that of the *state appearance record* algorithm [16], but we track colors instead of states. The intuition is that any cycle r' of $\mathcal{A}'$ corresponds to a cycle r of $\mathcal{A}$. If the union of the colors visited by r is R, then all the states in r' necessarily have all colors of R to the front of their history, there will be at least one transition t of r' for which the number of colors rotated by $\widetilde{\mathcal{U}}$ is $|R|$, and

for all the other transitions this number will be lesser or equal. Therefore, the color $2|R| + [R \not\models \alpha]$ selected for this transition t will be the highest of $\mathsf{Rep}(r')$ and will cause r' to be accepting iff r is accepting.

Note that this construction may produce $|Q| \times n!$ states in the worst case.

Example 2. The CAR arrow at the top-right of Figure 2 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 $\widetilde{\mathcal{U}}(\langle 0, 2, 1\rangle, \{2\})$ 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 $(0_1, \langle 2, 0, 1\rangle)$. Two colors $R = \{0, 2\}$ have been moved in the history by this transition, and since $R \models \alpha$ the transitions is labeled by color $2 \times |R| + 0 = 4$. 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 $R = \{0\} \not\models \alpha$, that loop is labeled by $2 \times |R| + 1 = 3$.

3.2 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.

We define the set of *index appearance records* as the set $\Pi(\mathcal{I})$ of permutations of Rabin pair indices. The output states $Q^{\mathrm{IAR}} = Q \times \Pi(\mathcal{I})$ are equipped with such a record to track the history of indices of the Rabin pairs (p_i, r_i) in the order the colors p_i were last seen.

We update those IAR using a function $\mathcal{U} : \Pi(\mathcal{I}) \times \mathcal{I} \to \Pi(\mathcal{I})$, such that $\mathcal{U}(\sigma, i) = \langle i, \sigma(0), \sigma(1), \ldots, \sigma(j-1), \sigma(j+1), \ldots, \sigma(|\mathcal{I}|-1)\rangle$ where $j = \sigma^{-1}(i)$ is the position of the index i in σ. In other words, $\mathcal{U}(\sigma, i)$ moves i to the front of σ by rotating the first $j+1$ elements. This update function can be generalized to a set of indices recursively with $\widetilde{\mathcal{U}}(\sigma, \emptyset) = \sigma$ and $\widetilde{\mathcal{U}}(\sigma, \{i\} \cup I) = \widetilde{\mathcal{U}}(\widetilde{\mathcal{U}}(\sigma, i), I)$.

When processing a transition labeled by colors $C \subseteq M$, we need to update the history for all indices $P(C) = \{i \in \mathcal{I} \mid p_i \in C\}$ of a prohibited color.

This construction builds an automaton with *parity max odd* acceptance over the color $M' = \{0, 1, \ldots, |\mathcal{I}|+2\}$. The transitions $\delta^{\mathrm{IAR}} \subseteq Q^{\mathrm{IAR}} \times \Sigma \times 2^{M'} \times Q^{\mathrm{IAR}}$ of the output automaton can be defined as:

$$\delta^{\mathrm{IAR}} = \left\{ (q, \sigma) \xrightarrow{x, \{c\}} (q', \pi) \;\middle|\; \begin{array}{l} q \xrightarrow{x, C} q' \in \delta,\ \pi = \widetilde{\mathcal{U}}(\sigma, P(C)),\ m = \mathcal{M}(\sigma, C), \\ c = 2m + 1 + [m \geq 0 \wedge p_{\sigma(m)} \in C] \end{array} \right\}$$

where $\mathcal{M}(\sigma, C) = \max\left(\{-\frac{1}{2}\} \cup \left\{i \in \{0, 1, |\mathcal{I}|-1\} \;\middle|\; p_{\sigma(i)} \in C \vee r_{\sigma(i)} \in C\right\}\right)$ is the rightmost index of σ corresponding to a pair with a color in C, or $-\frac{1}{2}$ if no such index exists.

Theorem 2 ([15]). *For any TELA $\mathcal{A} = (Q, M, \Sigma, \delta, q_0, \alpha)$ over the marks $M = \{0, \dots, n-1\}$ and such that α is a Rabin condition, there exists an equivalent TELA $\mathcal{A}' = (Q^{IAR}, M', \Sigma, \delta^{IAR}, (q_0, \pi_0), \alpha')$ where α' is a parity max odd acceptance formula over $2n+2$ colors. The initial permutation π_0 can be chosen arbitrarily. A dual construction transforms Streett into parity max even.*

For proof, we refer the reader to Löding [16] (for state-based acceptance) and to Křetínský et al. [15] (who adapted it to TELA).

For the intuition behind the definition of c in δ^{IAR}, imagine a transition $(q, \sigma) \xrightarrow{x,\{c\}} (q', \pi)$ on a cycle r' of $\mathcal{A}'$, and a matching transition $q \xrightarrow{x,C} q'$ from $\mathcal{A}$. Assume the corresponding cycle r of the input automaton visits all colors in $C' = \mathsf{Rep}(r)$. Because they are on the cycle r', the IARs σ, π, and the others on that cycle have all their indices $P(C')$ to the left of the permutation. When we scan the IAR σ from the right to the left to find the maximal index $m = \mathcal{M}(\sigma, C)$ corresponding to a pair matching C, three situations can occur: (1) If $m \geq 0$ and $p_{\sigma(m)} \in C$, we know that we are in the left part, and that all Rabin pairs of indices $\sigma(0), \dots, \sigma(m)$ are not satisfied on this cycle: the transition is labeled with $c = 2m + 2$ to indicate so. (2) If $m \geq 0$ and $p_{\sigma(m)} \notin C$, it *may* be the case that m is in the right part of the IAR, meaning that the Rabin pair of index $\sigma(m)$ is satisfied. We label the transition with $c = 2m+1$ to indicate acceptance, but this might be canceled by a another transition emitting a higher even value if $p_{\sigma(i)}$ appears elsewhere on this cycle. (3) Finally $m = -\frac{1}{2}$ occurs when $C = \emptyset$, in this case the transition is labeled by $c = 0$ as no pair is satisfied.

This procedure generates an automaton with $|Q| \times |\mathcal{I}|!$ states in the worst case, but unless colors occur multiple times in α, we usually have $|\mathcal{I}| \leq n/2$, making IAR preferable to CAR.

Example 3. The arrow IAR in Figure 2 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 $p_1 = 0 \in C$, this justifies that the output transition is labeled by $2m+1+1 = 4$.

3.3 Partial Degeneralization

We now define the partial degeneralization of a TELA A according to some 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 $\alpha[\bigwedge_{d \in D} \mathsf{Inf}(d) \leftarrow \mathsf{Inf}(e)][\bigvee_{d \in D} \mathsf{Fin}(d) \leftarrow \mathsf{Fin}(e)]$.

The construction ensures that the runs of the output that see all colors of D infinitely often also see e infinitely often. To do that, we consider an ordering of $\{d_0, d_1, \dots d_{|D|-1}\}$ of D, and equip each state of the output automaton by a *level* in $L = \{0, 1, \dots |D| - 1\}$. We jump from level i to 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 down to level 0 when a transition t leaving a state at level $|D|-1$ is labeled by $d_{|D|-1}$; moreover, since any cycle going through t will have seen all colors in D, we can add the new color e to t.

An optimization, commonly done in degeneralization procedures [11, 1], is that transition labeled by multiple consecutive colors of D may skip several levels. Let us define this *skipping* of levels more formally as a function $\mathcal{S} : L \times 2^M \to L \times 2^{\{e\}}$ that takes a level i and a set C of colors seen by some transition, and returns the new level j and a subset that is either $\emptyset$ or $\{e\}$ to indicate whether the new color should be added to the output transition.

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

$$j = \max\big(k \in \{i, i{+}1, \ldots, i{+}|D|\} \big| \{d_i, d_{(i+1) \bmod |D|}, \ldots, d_{(k+|D|-1) \bmod |D|}\} \subseteq C\big).$$

Theorem 3. *Let $\mathcal{A} = (Q, M, \Sigma, \delta, q_0, \alpha)$ be a TELA, and let $C \subseteq M$ be a set of marks. Let $D = \{d_0, d_1, \ldots, d_{|C|-1}\}$ be some ordering of the colors of C, and let $L = \{0, 1, \ldots, |C|\}$ be a set of* levels.

$\mathcal{A}$ is equivalent to its partial degeneralization according to C, defined by automaton $\mathcal{A}' = (Q', M', \Sigma, \delta', (q_0, i_0), \alpha')$ 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 $i_0 \in L$.

First, note that this procedure does not remove any color from the automaton. This is because 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 as a subsequent step, and this is done in our implementation.

Moreover, because the algorithm keeps all used colors, 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.

Finally, in the case where α is a generalized Büchi condition over the marks M, and $D = M$, then the resulting α' will be $\mathsf{Inf}(e)$, and removing all the now useless original colors will have the same effect as a classical degeneralization. In this sense, the degeneralization is a special case of the partial degeneralization. Similarly, this procedure can also be seen as a generalization of the transformation of generalized-Rabin automata into Rabin automata [12].

Cf. Appendix D

Example 4. In Figure 2, the arrow $\mathrm{PD}_{\{1,3\}}$ denotes the application of a partial degeneralization according to the set $M = \{1, 3\}$. This allows to rewrite acceptance'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 $d_0 = 3$, $d_1 = 1$.

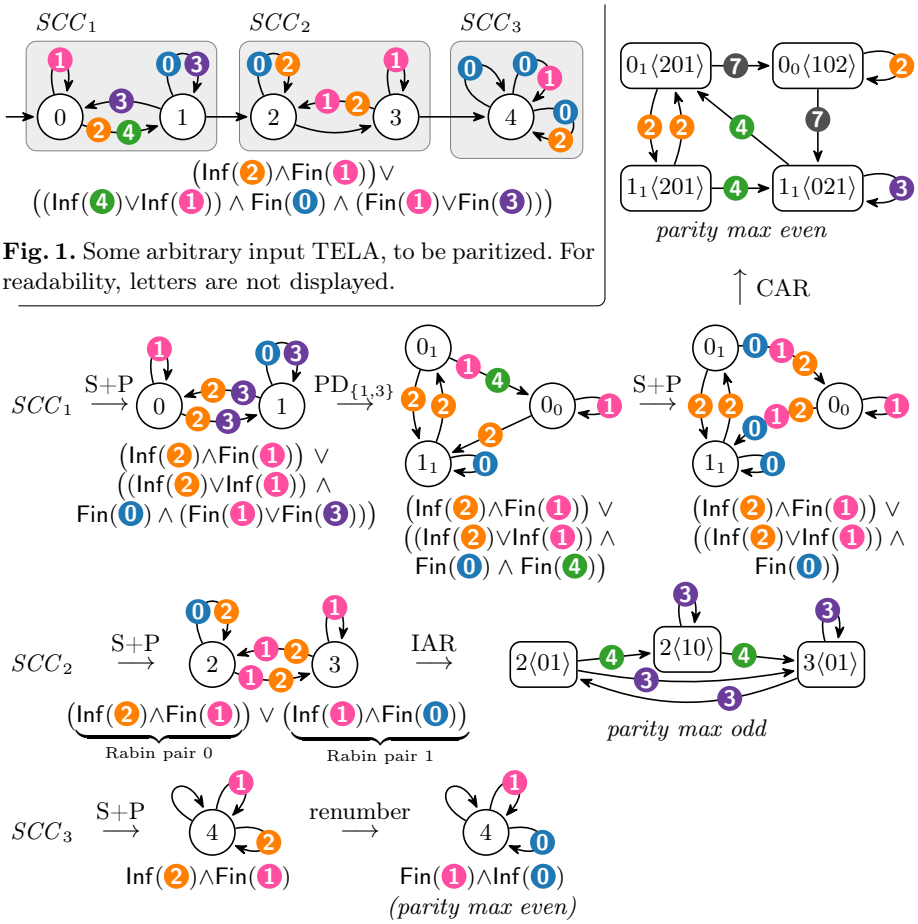


Fig. 1. Some arbitrary input TELA, to be paritized. For readability, letters are not displayed.

Fig. 2. Intermediate steps of the construction, handling the SCCs in different ways. These steps are explained at various places through Sections 3 and 4.

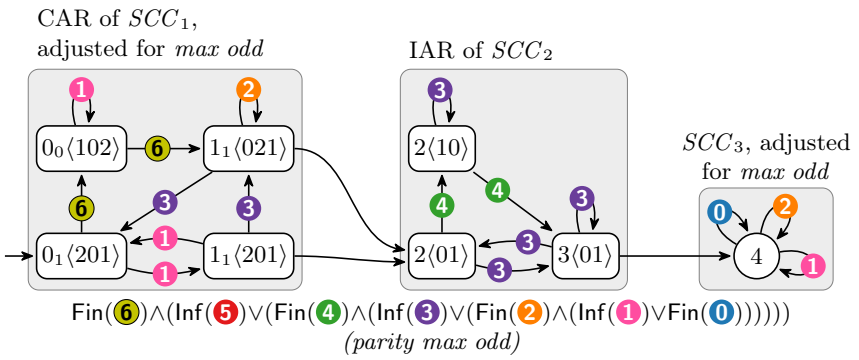
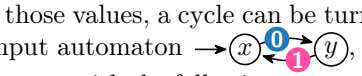
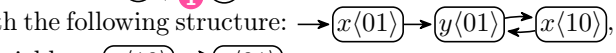


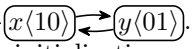
Fig. 3. Paritization of the automaton of Figure 1, combining the transformed SCCs of Figure 2 after adjustment to a common acceptance condition.

3.4 Optimizations

We now describe several optimizations for the aforementioned constructions.

Jump to bottom: The choice of the initial permutation π_0 in the CAR, in the IAR, or of the initial level i_0 in the partial degeneralization is arbitrary. With a bad selection of those values, a cycle can be 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 

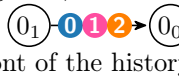
.

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 (in 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 [15]. Heuristics used in degeneralization algorithms to select initial level upon entering a new SCC [1] are then unnecessary.

History reuse: When processing an input transition labeled with multiple colors, the insertion order of those colors (resp. Rabin pair indices) in front of the history during an update of the CAR (resp. IAR) is arbitrary. Křetínský et al. [15] 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 2, this is how 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 [15]. 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.

4 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) Simplify its acceptance condition by removing unused colors ($\mathsf{Fin}(i)$ becomes $\top$, and $\mathsf{Inf}(i)$ becomes $\bot$ for any color i unused in $\mathcal{A}_{|S_i}$), or dually, colors that appear everywhere. Colors that always appear together can be replaced by a single color, and disjunctions of Inf or conjunctions of Fin can be reduced as discussed in Remark 2. Cf. Appendix B
   (c) Propagate colors in the SCC (Section 3.4).
   (d) 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 constructionwhich 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 1b as the acceptance changed. Cf. Appendix C
   (e) Transform the automaton $\mathcal{A}_{|S_i}$ into a *parity max* automaton R_i using the first applicable procedure from the following list:
      – If $\mathscr{L}(\mathcal{A}_{|S_i})$ is empty [3], 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.);

- 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. [14]); 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.
- 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 [5].

Figures 1–3 show this algorithm at work on a small example with three SCCs. Figure 3 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 2. The notation S+P refers to the Simplification of the acceptance condition (step 1b) and the Propagation of colors in the SCC (step 1c). On SCC_1, step 1b replaces $\mathsf{Inf}(4)$ by $\mathsf{Inf}(2)$, because these always occur together, and step 1c adds $\mathsf{Inf}(2)$ 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 $\mathsf{Inf}(4)$ by $\mathsf{Inf}(0)$ in the automaton, and after that the marks on the transitions before and after state 0_0 are propagated by step 1c. 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 $\mathsf{Inf}(3)$ and $\mathsf{Inf}(4)$ 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 3, 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.

5 Experimental Evaluation

The simple CAR described in Section 3.1, without the optimizations of Section 3.4 was implemented in Spot 2.8 [7] 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 [21] 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 [6] with specialized constructions for subclasses of LTL [9], or a Safra-based procedure [23]), 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 $\mathcal{A}_{\varphi_i}$.

In Spot 2.9, `to_parity()` was changed to implement Section 4 and the optimizations of Section 3.4. 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'2017. Among those automata, we have removed those that already had a parity acceptance (usually Büchi acceptance). The remaining set contains 32 automata with a generalized-Büchi condition, and 84 with a condition that mixes Fin and Inf terms (only 1 of these can be considered Rabin-like or Streett-like). The average number of accepting SCCs is 1.9 (min. 1, med. 1, max. 4). The average number of states is 46 (min. 1, med. 13, max. 986).

randltl contains 273 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. 9). The average number of accepting SCCs is 1.7 (min. 1, med. 1, max. 5). The average number of

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

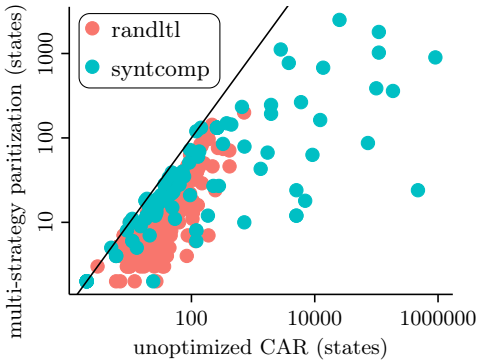


Fig. 4. Comparison of the new multi-strategy paritization (Section 4) against the unoptimized CAR (Section 3.1)

Table 2. Effect of disabling different optimizations on the arithmetic and geometric means of the number of states on both benchmarks.

configuration	amean	gmean
all	48.71	14.43
all − Rabin to Büchi	48.72	14.45
all − parity prefix	48.97	14.54
all − simplify acc	49.32	15.07
all − hist. reuse	51.01	15.18
all − reuse latest	51.05	15.29
all − propagate colors	55.69	16.91
all − partial degen	2165.50	20.20
unoptimized CAR	5375.02	45.16

states is 5.8 (min. 1, med. 4, max. 41). Only 13 of these automata have a Rabin-like or Streett-like acceptance condition.

The improvement of our new paritization based on multiple strategies over our old unoptimized CAR implementation is shown on Figure 4.

Cf. Appendix E

Table 2 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 1e. *Parity prefix* is the optimization mentioned at the very end of Section 4. *Simplify acc*, *propagate colors*, and *partial degen* correspond respectively to steps 1b and 1c, and 1d. 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 3.4. *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.

To assert the effect of the improved paritization on `ltlsynt`, we ran the entire SyntComp'17 benchmark (including formulas omitted before) with a timeout of `100 seconds`, and counted the number of cases solved by different configurations of `ltlsynt`, as reported in Table 3. We can see that improving CAR with all the tricks of Section 4 allowed the `ltlsynt`'s LAR-based approach to perform better than `ltlsynt`'s Safra-based approaches.

6 Conclusion

We have presented a procedure that converts any TELA into a transition-based parity automaton. Our algorithm combines algorithms that are transition-based

Table 3. Number of SyntComp'17 cases solved by `ltlsynt` under different configurations, with a timeout of 100 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=lar.old`	LTL to determ. TELA, then CAR of Section 3.1	175	7262s	
`--algo=sd`	LTL to Büchi, then split input/output variables, then Safra-based determinization [20]	177	6879s	
`--algo=ds`	LTL to Büchi, then Safra-based determinization, then split input/output variables [20]	180	6671s	
`--algo=lar`	LTL to determ. TELA, then approach of Section 4	185	6296s	Cf. Appendix F

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.

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 [15], detection of Büchi-realizable SCCs [14], detection of empty SCCs [3], 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 3.1 was first implemented in Spot by Maximilien Colange.

References

1. 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.
2. 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/.

3. 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.
4. R. Bloem, K. Chatterjee, and B. Jobstmann. *Graph Games and Reactive Synthesis*, chapter 27, pp. 921–962. Springer, 2018.
5. O. Carton and R. Maceiras. Computing the Rabin index of a parity automaton. *Informatique théorique et applications*, 33(6):495–505, 1999.
6. A. Duret-Lutz. LTL translation improvements in Spot 1.0. *International Journal on Critical Computer-Based Systems*, 5(1/2):31–54, 2014.
7. 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.
8. 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.
9. 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.
10. B. Farwer. *ω-Automata*, *LNCS* 2500, chapter 1, pp. 3–20. Springer, 2001.
11. P. Gastin and D. Oddoux. Fast LTL to Büchi automata translation. In *CAV'01*, *LNCS* 2102, pp. 53–65. Springer, 2001.
12. J. Křetínský and J. Esparza. Deterministic automata for the (F,G)-fragment of LTL. In *CAV'12*, *LNCS* 7358, pp. 7–22. Springer, 2012.
13. 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.
14. 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.
15. 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.
16. 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.
17. C. Löding. Optimal bounds for transformations of omega-automata. In *FSTTCS'99*, *LNCS* 1738, pp. 97–109. Springer, 1999.
18. 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.
19. 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.
20. 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`.
21. D. Müller and S. Sickert. LTL to deterministic Emerson-Lei automata. In *GandALF'17*, vol. 256 of *EPTCS*, pp. 180–194, 2017.
22. N. Piterman. From nondeterministic büchi and streett automata to deterministic parity automata. *Logical Methods in Computer Science*, 3(3), 2007.
23. R. Redziejowski. An improved construction of deterministic omega-automaton using derivatives. *Fundamenta Informaticae*, 119(3-4):393–406, 2012.
24. S. Safra and M. Y. Vardi. On ω-automata and temporal logic. In *STOC'89*, pp. 127–137. ACM, 1989.

The following appendices contain extra material for interested readers and are not part of the ATVA'20 proceedings. Appendices A to E were included in the ATVA'20 submission, to be read at the discretion of the reviewers. Appendix F was added while preparing the final version to answer one reviewer's question, unfortunately we did not have enough space to include it in the main text.

A Parity Acceptance Conditions

The following table gives *parity max odd* and *parity max even* acceptance formulas for various number of colors, using the HOA syntax [2]. This may help clarify corner cases for those formulas (e.g. with 0 or 1 color), or provide alternative interpretations as other classical acceptance conditions for cases with few colors. For Rabin(-like) and Streett(-like), the number of pairs is specified between parentheses (see Remark 1 on page 4).

	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 the HOA format, and therefore the automata we use, can label a transition with any number of colors. Of course, when the acceptance condition is a *parity max* condition, transitions with multiple colors can be simplified by removing all colors but the maximum. Moreover, the absence of colors behaves as an imaginary color -1 in terms of a *parity max* acceptance. Applications require each transition to feature exactly one color may simply increment all existing colors, introduce $\mathsf{Inf}(0)$ on uncolored transitions, and toggle the parity.

B Acceptance Simplifications

In Step 1b of the algorithm of Section 4, we suggest simplifying the acceptance condition using several rules, but do not list all the rules we apply. This appendix provides more details about the rules we use to simplify an acceptance condition.

As the CAR complexity is factorial in the number of colors, we are only interested in simplifications that reduce the number of colors, or that reduce the size of the acceptance formula without introducing more colors.

Assume that $\mathcal{A}$ is an automaton with acceptance condition α. As in Section 3.3, we write $\alpha[\beta \leftarrow \gamma]$ to mean "in α replace any subformula equal to β by γ".

Basic cleanup
If 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 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 complementary colors
If two colors i and j, are complementary[2], i.e. i is present on a transition iff j is not, then α can go through to 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 a conjunctions 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 simplified to $\beta[\mathsf{Inf}(i) \leftarrow \bot][\mathsf{Fin}(i) \leftarrow \top]$. Similarly, in a subformula of the form $\mathsf{Fin}(i) \vee \beta$ or $\mathsf{Inf}(i) \wedge \beta$, the subformula β can be simplified to $\beta[\mathsf{Inf}(i) \leftarrow \top][\mathsf{Fin}(i) \leftarrow \bot]$.

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 color k on 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 colors $m_1, \ldots, m_n$ occur only once in α, the general rule implemented consists in substituting any subformula $\mathsf{Inf}(j) \vee \bigwedge_{i=1}^{n}(\mathsf{Inf}(m_i) \vee \beta_i)$ by a subformula $\bigwedge_{i=1}^{n}(\mathsf{Inf}(m_i) \vee \beta_i)$, 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 $\mathsf{Inf}(j)$ is first distributed inside the $\bigwedge_{i=1}^{n}$ before applying the trick described in the previous paragraph.
A dual rule allows to remove $\mathsf{Fin}(j)$ in $\mathsf{Fin}(j) \wedge \bigvee_{i=1}^{n}(\mathsf{Fin}(m_i) \wedge \beta_i)$.

Basic cleanup, merging colors, and simplifying complementary colors where already implemented in Spot. Unit propagation and fusing were added while working on this paritization procedure, and especially while looking at the effect of the algorithm on automata with random acceptance conditions.

[2] Those rules could be generalized to cases where each transition contain i, j, or both. https://gitlab.lrde.epita.fr/spot/spot/-/issues/403

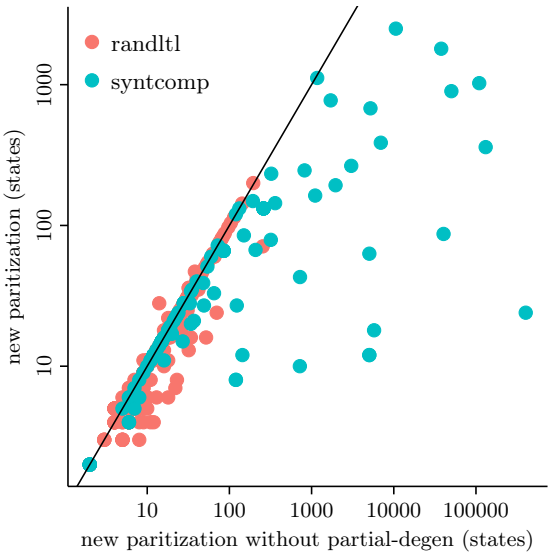


Fig. 5. Effect of disabling the partial degeneralization in the new paritization.

C 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 2 (page 9), CAR would have tracked four colors and would have built a 6-state automaton.

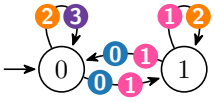
Another argument in favor of doing a degeneralization is that it may help shape the acceptance condition into something that is easier to paritize. As an example, it may allow us to use IAR instead of CAR. From this perspective, it can be useful to use a partial degeneralization even if it does not reduce 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 3.4), it may be the case that it fails to find a good ordering, or that there is no good order for the entire automaton.

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 5.

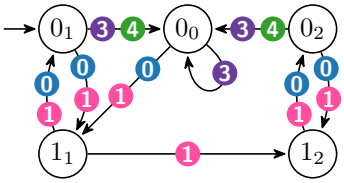
The following automaton illustrates a case where partial degeneralization produces an automaton larger than direct application of CAR.

Consider the following input automaton:



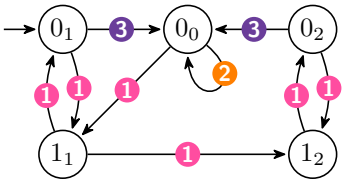
$(\mathsf{Fin}(3) \wedge (\mathsf{Inf}(1) \vee \mathsf{Fin}(0))) \vee (\mathsf{Inf}(0) \wedge \mathsf{Inf}(2) \wedge \mathsf{Inf}(3))$

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 $d_2 = 3$.)



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

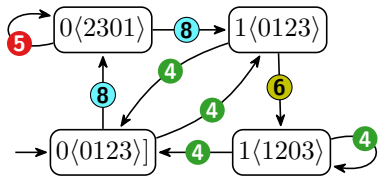
By chance, the shape of the acceptance formula corresponds to a *parity max odd* formula, up to a renaming of colors. After renaming 3, 4 to 2, 3, and keeping only the maximum color on each edge, we end up with a parity automaton:



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

For that reason, our implementation features an option to try the paritization with and without partial degeneralization. It was however disabled that in the presented experiments, as it would hinder the interpretation of the results.

Note that this specific automaton can be fixed by implementing[3] 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.

D Partial Degeneralization vs. Degeneralization and Co

For size reasons, we did not discuss in details how the partial degeneralization of Section 3.3 differs from the classical degeneralization, or how it relates to the transformation of generalized-Rabin to Rabin [12].

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). An improvement is to jump multiple levels at one time [11] when possible, but never going past the last level, since that is where acceptance states are located.

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 $\mathcal{S}(s, C)$ (page 8) 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.

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.

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. [12] can be seen 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.

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

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) and be used in other cases (in our case, preprocessing TELA with arbitrary acceptance conditions before applying CAR or IAR).

E More Detailed Comparison of Optimizations

Table 4 is a more detailed version of Table 2 where we can see the effect on the two datasets, and we also consider the case where we disable all optimizations or only enable one at a time. 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 4 compares the cases summarized by lines *all* and *unoptimized CAR*, while Figure 5 plots the data of lines *all* and *all − partial degen*.

Table 4. Effect of various optimizations on the paritization procedure. Configuration "all" corresponds to the algorithm of section 4. Configuration "unoptimized CAR" corresponds to the basic CAR implementation of section 3.1. Configuration "nothing" implements only the SCC-based selection of CAR or IAR.

dataset	randltl		syntcomp		both	
config	amean	gmean	amean	gmean	amean	gmean
all	17.06	10.68	124.5	29.72	48.71	14.43
all − Rabin to Büchi	17.08	10.70	124.5	29.72	48.72	14.45
all − parity prefix	17.44	10.79	124.5	29.72	48.97	14.54
all − simplify acc.	17.93	11.35	124.5	29.72	49.32	15.07
all − hist. reuse	18.52	11.37	128.8	30.30	51.01	15.18
all − reuse last	18.60	11.50	128.8	30.28	51.05	15.29
all − propagate colors	25.18	13.33	128.8	29.85	55.69	16.91
nothing + partial degen	33.19	17.24	144.2	30.54	65.89	20.40
all − partial degen	18.76	11.78	7306.4	73.60	2165.50	20.20
nothing + propagate colors	26.34	16.95	9351.7	96.22	2773.34	28.27
nothing + hist. reuse (last)	30.22	17.31	15244.3	91.02	4511.89	28.23
nothing + hist. reuse (first)	35.53	20.81	17101.3	101.21	5062.67	33.16
nothing + simplify acc.	34.26	18.49	17433.1	102.23	5159.51	30.60
nothing + parity prefix	35.91	20.67	17454.7	103.29	5167.02	33.20
nothing + Rabin to Büchi	36.56	21.13	17454.7	103.29	5167.47	33.72
nothing	36.57	21.14	17454.7	103.29	5167.48	33.73
unoptimized CAR	49.64	29.37	18127.9	126.57	5375.02	45.16

F Differences Between `ltlsynt` Configurations

While table 3 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 5 reveals that despite the number of solved cases increasing, some cases are solved by other configurations but not by the new LAR.

The case of `round_robin_arbiter_5.tlsf` where `lar` fails and `lar.old` could be a measurement error. While preparing the camera-ready version of this article, we could not reproduce the failure of the `lar` configuration on this specification: `lar` and `lar.old` both seem to take 54 seconds (way below the 100-second timeout), and in both cases 53 seconds are spent in the translation, and 1 second in the paritization.

In a future work, we plan to update this benchmark with data from the SyntComp'20 edition.

Table 5. Specifications of the SyntComp'17 benchmark for which at least one configuration of `ltlsynt` succeeded (✓) and at least one configuration failed (−).

file	`lar.old`	`sd`	`ds`	`lar`
`ltl2dba_U1_6.tlsf`	✓	−	✓	✓
`loadfull5.tlsf`	✓	−	✓	✓
`amba_decomposed_arbiter.tlsf`	−	✓	✓	✓
`full_arbiter_4.tlsf`	−	✓	✓	✓
`prioritized_arbiter_4.tlsf`	−	✓	✓	✓
`round_robin_arbiter_6.tlsf`	−	✓	✓	✓
`detector_6.tlsf`	−	−	−	✓
`detector_unreal_8.tlsf`	−	−	−	✓
`detector_unreal_10.tlsf`	−	−	−	✓
`detector_unreal_12.tlsf`	−	−	−	✓
`lilydemo18.tlsf`	−	−	−	✓
`ltl2dba08.tlsf`	−	−	−	✓
`ltl2dba_C2_6.tlsf`	−	−	−	✓
`round_robin_arbiter_5.tlsf`	✓	✓	✓	−
`full_arbiter_5.tlsf`	−	−	✓	−