Practical Applications of the Alternating Cycle Decomposition

Antonio Casares[1] (✉), Alexandre Duret-Lutz[2], Klara J. Meyer[3], Florian Renkin[2], and Salomon Sickert[4]*

[1] LaBRI, Université de Bordeaux, France, antonio.casares-santos@labri.fr
[2] LRDE, EPITA, France, adl@lrde.epita.fr, frenkin@lrde.epita.fr
[3] Independent Researcher, email@klarameyer.de
[4] School of Computer Science and Engineering, The Hebrew University, Israel, salomon.sickert@mail.huji.ac.il

Abstract. In 2021, Casares, Colcombet, and Fijalkow introduced the Alternating Cycle Decomposition (ACD) to study properties and transformations of Muller automata. We present the first practical implementation of the ACD in two different tools, Owl and Spot, and adapt it to the framework of Emerson-Lei automata, i.e., ω-automata whose acceptance conditions are defined by Boolean formulas. The ACD provides a transformation of Emerson-Lei automata into parity automata with strong optimality guarantees: the resulting parity automaton is minimal among those automata that can be obtained by duplication of states. Our empirical results show that this transformation is usable in practice. Further, we show how the ACD can generalize many other specialized constructions such as deciding typeness of automata and degeneralization of generalized Büchi automata, providing a framework of practical algorithms for ω-automata.

1 Introduction

Automata over infinite words have many applications, including verification and synthesis of reactive systems with specifications given in formalisms such as *Linear Temporal Logic* (LTL) [27, 23, 11, 12, 2, 29]. The synthesis problem from LTL specifications asks, given an LTL formula φ, to build a controller that processes an input word letter by letter, producing an output word, such that the combined input-output-word satisfies φ. The automata-theoretic approach to this problem (first introduced by Pnueli and Rosner [27]) consists of building a deterministic ω-automaton $\mathcal{A}$ equivalent to the LTL specification φ, then construct a game from $\mathcal{A}$ in which the opponent chooses the input letters for the automaton, and finally solve this game and obtain a controller from a winning strategy (whenever such a strategy exists). The automaton $\mathcal{A}$ can use different kinds of acceptance conditions (Rabin, Emerson-Lei, Muller, parity...) and

* Salomon Sickert is supported in part by the Deutsche Forschungsgemeinschaft (DFG) under project number 436811179, and in part funded by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme under grant agreement No. 787367 (PaVeS)

thus we obtain games with different winning conditions. Among these games, parity games are the easiest to solve and there are highly-developed techniques for parity games solvers. Thus it is common practice to transform the automaton $\mathcal{A}$ to a parity one (for which we might need to augment the state space of the automaton). The top-ranked tools in the SyntComp competitions [17], Strix [23] (winner in editions 2018, 2019, 2020 and 2021) and `ltlsynt` [26], use this approach, producing a transition-based Emerson-Lei automata (TELA) as an intermediate step before constructing the parity automaton. For this reason, optimal and efficient procedures to transform Emerson-Lei automata into parity automata are of great importance.

Emerson-Lei (EL) acceptance conditions (first defined by Emerson and Lei [10], and reinvented in the HOA format [3]) are arbitrary positive Boolean formulas over the primitives $\mathsf{Inf}(c)$ and $\mathsf{Fin}(c)$ where c's are *colors* from a set Γ. A run is accepting if the set of colors $\mathcal{F} \subseteq 2^{\Gamma}$ seen infinitely often is a satisfying assignment to the EL acceptance condition (see Section 2 for a formal definition). Note that an explicit representation of all satisfying assignments is comparable to the *Muller condition* [15, Section 1.3.2]. Since the Boolean structure of LTL formulas can be mimicked by the Emerson-Lei acceptance conditions, a translation of LTL formulas to Emerson-Lei automata is particularly convenient.

Many algorithms to transform Emerson-Lei and Muller automata to parity have been proposed. In essence they all transform an automaton by turning each original state q into multiple states of the form (q, r) where r records some information about the current run, and transitions leaving (q, r) otherwise have a one-to-one mapping with those leaving q. Definition 3 calls this a *locally bijective morphism*, and we like to refer to those as *algorithms that duplicate states*. For instance in the *Later Appearance Record (LAR)* [16], r is a list of all colors ordered by most recent appearance, producing therefore a blow-up of $|\Gamma|!$ in the state-space of the automaton. The *State Appearance Record (SAR)* [24, 22] is a variation of this idea for state-based conditions, and the *Color Appearance Record (CAR)* [28] is a variation for the Emerson-Lei condition. The *Index Appearance Record (IAR)* [24, 22, 20] is a specialized construction for Rabin and Streett conditions, where r is now an ordering of pair indices. These algorithms have no particular insights about the input acceptance condition, such as inclusion or redundancies between colors (or pairs). In the Zielonka-tree transformation [31], r is a reference to a branch in a tree representation of a Muller condition. That tree representation is tailored to the condition and allows such simplifications compared to previous methods (it can be proven to be always better [6, 25]). While none of these algorithms use the structure of the input automaton to optimize the produced automata, some heuristics have been proposed [28, 25, 21].

In 2021, inspired by the *Zielonka tree*, Casares et al. introduced the Alternating Cycle Decomposition (ACD) of a Muller automaton [6]. Simply put, the ACD is a forest, i.e., a list of trees, that captures how accepting and rejecting cycles interleave in the automaton. They use the ACD to transform Muller automata into parity automata, and they prove a strong optimality result: the resulting automaton uses an optimal number of colors and has a minimal number of states among those parity automata that can be obtained by duplicating states of the original one (see Theorem 1 for a formal statement). The main novelty of this transformation is that it does not only take into account the structure of both the acceptance condition and the automaton, but it exactly captures how they interact with each other. Moreover, Casares et al. [6] show that we can obtain some other valuable information about a Muller automaton from its ACD: for example the ACD can be used to decide *typeness*, i.e, if we can relabel it with another acceptance condition (parity, Rabin, Streett...). Their approach is primarily theoretical and puts the emphasis on how the ACD can be useful to obtain new results concerning Muller automata, but little is said about the costs of computing the ACD or the applicability of the transformation in practice.

Contributions. In this paper, we show that the ACD is practical. We adapt the definition of the ACD to Emerson-Lei automata and the HOA format [3]. We implement the ACD and the associated transformation in two tools: Owl [18] and Spot [9], providing baselines for efficient implementations of these structures. We show that the ACD gives a usable and useful method to transform Emerson-Lei automata into parity ones, improving upon any previous transformation in terms of the size of the output parity automaton. We extend the ACD to produce state-based automata, and show that the ACD generally beats traditional degeneralization-based procedures. Our implementation can also use the ACD to check typeness of deterministic automata.

Structure of the paper. We begin by providing some common definitions in Section 2. In Section 3, we define the Alternating Cycle Decomposition, adapting the definition of Casares et al. [6] to Emerson-Lei automata, and we provide an algorithm to compute it. In Section 5, we study the transformation of Emerson-Lei automata into parity ones using the ACD and we show experimental results obtained by comparing the ACD-transform implemented in Spot and Owl with other commonly used transformations. In Section 6 we show experimental results in the particular case of degeneralization of generalized Büchi automata. In Section 7 we discuss the utility of the ACD to decide typeness of automata.

2 Preliminaries

We denote by $|A|$ the cardinality of a set A and by 2^A its power set. For a finite alphabet Σ, we write Σ^* and Σ^ω for the sets of finite and infinite words, respectively, over Σ. The empty word is denoted by ε. Given $v \in \Sigma^*, w \in \Sigma^\omega$, we denote their concatenation by $v \cdot w$ and we write $v \sqsubseteq w$ if v is a prefix of w. We note $\inf(w)$ the set of letters that occur infinitely often in w. Given a map $\sigma \colon A \to B$ and a subset $A' \subseteq A$, we denote $\sigma|_{A'}$ the restriction of σ to A'. We extend σ to A^* and A^ω component-wise and we denote these extensions by σ whenever no confusion arises.

A (directed, edge-colored) *graph* is a pair $G = (V, E)$ where V is a finite set of vertices and $E \subseteq V \times \Gamma \times V$ is a finite set of Γ-colored edges. Note that with

Table 1: Encoding of common acceptance conditions into Emerson-Lei conditions. The variables $c, c_0, c_1, \dots$ stand for arbitrary colors from the set $\mathit{\Gamma}$.

(B)	Büchi	$\mathsf{Inf}(c)$
(GB)	generalized Büchi	$\bigwedge_i \mathsf{Inf}(c_i)$
(C)	co-Büchi	$\mathsf{Fin}(c)$
(GC)	generalized co-Büchi	$\bigvee_i \mathsf{Fin}(c_i)$
(R)	Rabin	$\bigvee_i (\mathsf{Fin}(c_{2i}) \wedge \mathsf{Inf}(c_{2i+1}))$
(S)	Streett	$\bigwedge_i (\mathsf{Inf}(c_{2i}) \vee \mathsf{Fin}(c_{2i+1}))$
(P)	parity min even	$\mathsf{Inf}(0) \vee (\mathsf{Fin}(1) \wedge (\mathsf{Inf}(2) \vee (\mathsf{Fin}(3) \wedge \dots)))$
	parity min odd	$\mathsf{Fin}(0) \wedge (\mathsf{Inf}(1) \vee (\mathsf{Fin}(2) \wedge (\mathsf{Inf}(3) \vee \dots)))$

this definition one can have multiple differently colored edges from a vertex v to a vertex u. A graph $G' = (V', E')$ is a *subgraph* of G (written $G' \subseteq G$) if $V' \subseteq V$ and $E' \subseteq E$. A graph $G = (V, E)$ is *strongly connected* if for every pair of vertices $(v, u) \in V^2$ there is a path from v to u. A *strongly connected component* (SCC) of a graph G is a maximal strongly connected subgraph of G.

Emerson-Lei acceptance conditions. Let $\mathit{\Gamma} = \{0, \dots, n-1\}$ be a finite set of n integers called *colors*, from now on also written $\mathit{\Gamma} = \{$⓿, ❶, $\dots\}$ in our examples. We define the set $\mathbb{EL}(\mathit{\Gamma})$ of *acceptance conditions* according to the following grammar, where c stands for any color in $\mathit{\Gamma}$:

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

Acceptance conditions are interpreted over subsets of $\mathit{\Gamma}$. For $C \subseteq \mathit{\Gamma}$ we define the satisfaction relation $C \models \alpha$ inductively according to the following semantics:

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

We denote by $\neg\alpha$ the negation of the acceptance condition α, i.e., $\mathsf{Fin}(m)$ becomes $\mathsf{Inf}(m)$, and vice-versa, $\wedge$ becomes $\vee$, etc. We assume that constants are propagated, i.e., a formula is either $\top$, $\bot$, or does not contain $\top$ and $\bot$.

Table 1 shows how common acceptance conditions can be encoded into Emerson-Lei conditions. Note that colors may appear multiple times; for instance $(\mathsf{Fin}($⓿$) \wedge \mathsf{Inf}($❶$)) \vee (\mathsf{Fin}($❶$) \wedge \mathsf{Inf}($⓿$))$ is a Rabin condition.

Emerson-Lei automata. A *transition-based Emerson-Lei automaton* (TELA) is a tuple $\mathcal{A} = (Q, \mathit{\Sigma}, Q_0, \mathit{\Delta}, \mathit{\Gamma}, \alpha)$, where Q is a finite set of states, $\mathit{\Sigma}$ is a finite input alphabet, $Q_0 \subseteq Q$ is a non-empty set of initial states, $\mathit{\Gamma}$ is a set of colors, $\mathit{\Delta} \subseteq Q \times \mathit{\Sigma} \times 2^{\mathit{\Gamma}} \times Q$ is a finite set of transitions, and $\alpha \in \mathbb{EL}(\mathit{\Gamma})$ is an Emerson-Lei condition. The *graph of* $\mathcal{A}$ is the directed edge-colored graph $G_{\mathcal{A}} = (Q, E)$ where the edges $E = \{(q, C, q') : \exists a \in \mathit{\Sigma}.\ (q, a, C, q') \in \mathit{\Delta}\}$ are obtained from $\mathit{\Delta}$ by removing $\mathit{\Sigma}$. We denote the transition $(q, a, C, q') \in \mathit{\Delta}$ and the edge $(q, C, q') \in E$ by $q \xrightarrow{a:C} q'$ and $q \xrightarrow{C} q'$, respectively. Further, we might omit a or C if they are clear from the context. We denote by γ the projection of Δ or E to the set of colors Γ. Given a word $w = a_0 \cdot a_1 \cdot a_2 \cdots \in \Sigma^\omega$, a *run over* w in $\mathcal{A}$ is a sequence $\varrho = (q_0, a_0, C_0, q_1) \cdot (q_1, a_1, C_1, q_2) \cdots \in \Delta^\omega$ such that $q_0 \in Q_0$. The *output* of the run ϱ, is the word $\gamma(\varrho) \in (2^\Gamma)^\omega$. A run ϱ is *accepting* if $\inf(\gamma(\varrho)) \models \alpha$. A word $w \in \Sigma^\omega$ is *accepted* (or *recognized*) by $\mathcal{A}$ if there exists an accepting run over w in $\mathcal{A}$. We denote $\mathcal{L}(\mathcal{A})$ the set of words accepted by $\mathcal{A}$. Two automata $\mathcal{A}$, $\mathcal{A}'$ are *equivalent* if $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$. The size of an automaton, written $|\mathcal{A}|$, is the cardinality of its set of states. A state $q \in Q$ is *reachable* if there is a path from some state in Q_0 to q in $G_\mathcal{A}$.

An automaton $\mathcal{A}$ is *deterministic* if Q_0 is a singleton and for every $q \in Q$ and $a \in \Sigma$ there is at most one transition from q labeled with a, $q \xrightarrow{a:C} q' \in \Delta$.

We will use automata with acceptance defined over transitions (instead of stated-based acceptance) by default. However, in Sections 5 and 6 we will also discuss transformations towards automata with state-based acceptance.

If the acceptance condition of an automaton is represented as a condition of kind X (cf. Table 1), we call it an X*-automaton.* We assume that each transition of a parity-automaton is colored with exactly one color; this can be achieved by substituting the set C in a transition $q \xrightarrow{a:C} q'$ by $\min C$ (if $C \neq \emptyset$) or by $\{|\Gamma|+1\}$ if $C = \emptyset$. (If C is a singleton we will omit the brackets in the notation).

Labeled trees. A *tree* is a non-empty prefix-closed set $T \subseteq \mathbb{N}^*$ whose elements are called *nodes*. It is partially ordered by the prefix relation; if $x \sqsubseteq y$ we say that x is an *ancestor* of y and y is a *descendant* of x (we add the adjective "strict" if moreover $x \neq y$). The empty string ε is the *root* of the tree. The set of *children* of a node $x \in T$ is $\mathit{Children}_T(x) = \{x \cdot i \in T \ : \ i \in \mathbb{N}\}$. The set of leaves of T is $\mathit{Leaves}(T) = \{x \in T \ : \ \mathit{Children}_T(x) = \emptyset\}$. Nodes belonging to a same set $\mathit{Children}_T(x)$ are called *siblings*, and they are ordered from left to right by increasing value of their last component. If A is a set of labels, an A*-labeled tree* is a pair $\langle T, \eta \rangle$ of a tree T and a map $\eta \colon T \to A$. The *depth* of a node x is $\mathit{Depth}(x) = |x|$. The *height* of T is $\mathit{Height}(T) = \max\limits_{x \in T} \mathit{Depth}(x)$.

3 The Alternating Cycle Decomposition

The *Alternating Cycle Decomposition* (ACD), proposed by Casares et al. [6], is a generalization of the Zielonka tree. The ACD of an automaton $\mathcal{A}$ is a forest, a collection of trees, labeled with accepting and rejecting cycles of the automaton. For each SCC of $\mathcal{A}$ we have a unique tree and the labeling of each tree alternates between accepting and rejecting cycles. Thus the ACD captures the complexity of the cycle structure of each SCC. We present now the definition of the ACD adapted to TELA.

For the rest of this section, let $\mathcal{A} = (Q, \Sigma, Q_0, \Delta, \Gamma, \alpha)$ be a TELA and let $G_\mathcal{A} = (Q, E)$ be the associated graph with edges colored by $\gamma \colon E \to 2^\Gamma$. We lift γ to sets and define $\gamma(E') = \bigcup\limits_{e \in E'} \gamma(e)$ for every subset $E' \subseteq E$.

Definition 1. *A* cycle *of $\mathcal{A}$ is a subset of edges $\ell \subseteq E$ forming a closed path in $G_{\mathcal{A}}$. A cycle ℓ is* accepting (resp. rejecting) *if $\gamma(\ell) \vDash \alpha$ (resp. $\gamma(\ell) \nvDash \alpha$). The set of* states *of a cycle ℓ is $\mathit{States}(\ell) = \{q \in Q : \text{some } e \in \ell \text{ passes through } q\}$. The set of cycles of $\mathcal{A}$ is denoted $\mathcal{Cycles}(\mathcal{A})$. It is (partially) ordered by set inclusion.*

Definition 2 ([6]). *Let $S_1, \dots, S_k$ be an enumeration of the strongly connected components of $G_{\mathcal{A}}$. The* Alternating Cycle Decomposition *of $\mathcal{A}$, denoted $\mathcal{ACD}(\mathcal{A})$, is a collection of k $\mathcal{Cycles}(\mathcal{A})$-labeled trees $\langle \mathcal{T}_1, \dots, \mathcal{T}_k \rangle$ with $\mathcal{T}_i = \langle T_i, \eta_i \rangle$ such that:*

- *$\eta_i(\varepsilon)$ is the set of edges of S_i, for $i = 1, \dots, k$.*
- *If $x \in T_i$ and $\eta_i(x)$ is an accepting cycle, then x has a child in $\mathcal{T}_i$ for each maximal element in $\{\ell \in \mathcal{Cycles}(\mathcal{A}) : \ell \subseteq \eta_i(x) \text{ and } \ell \text{ is rejecting}\}$. In this case, we say that x is a* round node.
- *If $x \in T_i$ and $\eta_i(x)$ is a rejecting cycle, then x has a child in $\mathcal{T}_i$ for each maximal element in $\{\ell \in \mathcal{Cycles}(\mathcal{A}) : \ell \subseteq \eta_i(x) \text{ and } \ell \text{ is accepting}\}$. In this case, we say that x is a* square node.

If $q \in Q$ is a state belonging to the SCC S_i in $\mathcal{A}$, we define the *tree associated to q* as the subtree $\mathcal{T}_q = \langle T_q, \eta_q \rangle$ given by:

$$T_q = \{\varepsilon\} \cup \{x \in T_i : q \in \mathit{States}(\eta_i(x))\}, \quad \eta_q = \eta_i|_{T_q}.$$

Remark 1. We provide examples online at https://spot.lrde.epita.fr/ipynb/zlktree.html and an executable copy of this notebook is included in the artifact [8].

4 An Efficient Computation of the ACD

In this section we give an algorithm to compute the Alternating Cycle Decomposition of an Emerson-Lei automaton $\mathcal{A}$, implemented in Owl [18] and Spot [9]. This can be done by first computing an SCC-decomposition of $\mathcal{G}_{\mathcal{A}}$ which gives us the labels of the roots of the trees $\langle \mathcal{T}_1, \dots, \mathcal{T}_k \rangle$, and then recursively computing the children of the nodes of each tree, following the definition of $\mathcal{ACD}(\mathcal{A})$. Algorithm 1 shows how to compute the children of a given node and uses notation we introduce now.

Let $C \subseteq \Gamma$ be a subset of colors and let $\mathcal{S} = (Q_S, E_S) \subseteq G_{\mathcal{A}}$ be a subgraph. We define the *projection of $\mathcal{S}$ on C*, denoted $\mathcal{S}_{\downarrow C} = (Q_S, E'_S)$, as the subgraph of $\mathcal{S}$ obtained by removing the edges $e \in E_S$ such that $\gamma(e) \nsubseteq C$, that is, $E'_S = \{(q, D, q') \in E_S : D \subseteq C\}$. We write $\mathit{Colors}(\mathcal{S}) = \bigcup_{e \in E_S} \gamma(e)$. We say that $\mathcal{S}' \subseteq \mathcal{S}$ is an *C-strongly connected component* in $\mathcal{S}$ (C-SCC) if it is an SCC of $\mathcal{S}$ and $\mathit{Colors}(\mathcal{S}') = C$. Further, $\max_{\subseteq}$ is the *set* of all maximal elements according to the partial order defined by $\subseteq$.

Note that Algorithm 1 uses Algorithm 2, which simplifies the Emerson-Lei conditions before passing the formula to a **Max-SAT** function (a SAT-solver that computes maximal satisfying assignments, e.g., by clause blocking) [4]. This preprocessing ensures that the ACD for Rabin or Streett acceptance conditions can be constructed without making use of the general purpose algorithm for computing maximal satisfying assignments.

Algorithm 1 Computing the children of a node.

```
1: Input: A cycle $\mathcal{S} = \eta_i(x)$ corresponding to the label of a node $x$ of $\mathcal{ACD}(\mathcal{A})$.
2: Output: The set of labels for the children of $x$, $(\mathcal{S}_1, \dots, \mathcal{S}_k)$.
3: function Compute-Children($\mathcal{S}$)
4:     $children \leftarrow \emptyset$, $C \leftarrow Colors(\mathcal{S})$
5:     if $C \vDash \alpha$ then        ▷ Maximal subsets $D \subsetneq C$ such that $D \vDash \alpha \Leftrightarrow C \nvDash \alpha$
6:         $\{C_1, \dots, C_k\} \leftarrow$ Max-Satisfying-Subsets$(C, \neg\alpha)$
7:     else
8:         $\{C_1, \dots, C_k\} \leftarrow$ Max-Satisfying-Subsets$(C, \alpha)$
9:     for $D \in \{C_1, \dots, C_k\}$ do
10:        for $\mathcal{S}' \in$ SCCs of $\mathcal{S}_{\downarrow D}$ do        ▷ These might not be $D$-SCC in $\mathcal{S}$
11:            if $Colors(\mathcal{S}') \vDash \alpha \;\Leftrightarrow\; D \vDash \alpha$ then
12:                $children \leftarrow children \cup \{\mathcal{S}'\}$
13:            else
14:                $children \leftarrow children \cup$ Compute-Children$(\mathcal{S}')$
15:    return $\max_{\subseteq} children$        ▷ Remove from children non-maximal cycles
```

Algorithm 2 The subprocedure `Max-Satisfying-Subsets`.

```
1: Input: A subset of colors $C \subseteq \Gamma$ and an EL condition $\alpha \in \mathbb{EL}(\Gamma)$.
2: Output: $\max_{\subseteq}\{D \subseteq C \;:\; D \vDash \alpha\}$.
3: function Max-Satisfying-Subsets$(C, \alpha)$
4:     if $C \vDash \alpha$ then
5:         return $\{C\}$
6:     $\alpha \leftarrow \alpha[\text{if } c \in C \text{ then } c \text{ else } \bot]$        ▷ Replace colors not in $C$ by false
7:     $L \leftarrow \{c \in C : \neg c \text{ does not occur in } \alpha\}$
8:     if $L \neq \emptyset$ then
9:         $\alpha \leftarrow \alpha[\text{if } c \in L \text{ then } \top \text{ else } c]$        ▷ Replace colors in $L$ by true
10:        $\{C_1, \dots, C_k\} \leftarrow$ Max-Satisfying-Subsets$(C \setminus L, \alpha)$
11:        return $\{C_1 \cup L, \dots, C_k \cup L\}$
12:    if $\alpha = \neg c_1 \vee \dots \vee \neg c_n$ then
13:        return $\{\{c_1, \dots, c_n\} \setminus \{c_i\} : 1 \leq i \leq n\}$
14:    return Max-SAT$(\alpha)$
```

Memoization. To optimize the construction of the ACD and to avoid duplicated recursive calls, we perform two kinds of memoization: First, we memoize the results of calling Algorithm 2 from Algorithm 1. (Thus we implicitly construct a Zielonka DAG for α.) Second, we memoize the recursive calls to Algorithm 1: this is useful, as distinct nodes in the ACD can be labeled by the same cycles.

5 From Emerson-Lei to Parity Automata

In this section we describe the transformation from TELA to parity automata using the Alternating Cycle Decomposition [6]. This transformation provides strong optimality guarantees: the resulting parity automaton has minimal size among those that can be produced without merging states from the TELA and it uses an optimal number of colors (Theorem 1). We also show that this transformation can be adapted to produce state-based automata. Note that in this case we loose the first optimality guarantee.

5.1 The ACD Transformation

Let $\mathcal{A} = (Q, \Sigma, Q_0, \Delta, \Gamma, \alpha)$ be a TELA and let $\mathcal{ACD}(\mathcal{A}) = \langle \mathcal{T}_1, \dots, \mathcal{T}_k \rangle$. We introduce the following notation that will allow us to move in the ACD.

Given a transition $e = q \xrightarrow{a:C} q'$ such that both q and q' belong to the i-th SCC of $\mathcal{A}$ and a node $x \in T_i$, we define $\mathit{Support}(x, e)$ to be the least ancestor z of x in $\mathcal{T}_i$ such that $e \in \eta_i(z)$. If $\mathit{Support}(x, e) \neq x$ and it is not a leaf in $\mathcal{T}_{q'}$, let z' be the only child of $\mathit{Support}(x, e)$ that is an ancestor of x, and let $y_1, \dots, y_s$ be an enumeration from left to right of the nodes in $\mathit{Children}_{\mathcal{T}_{q'}}(\mathit{Support}(x, e))$. We define $\mathit{NextBranch}(x, e)$ as:

$$\begin{cases} \mathit{Support}(x, e), & \text{if } \mathit{Support}(x, e) = x \text{ or if } \mathit{Support}(x, e) \text{ is a leaf in } \mathcal{T}_{q'}, \\ y_1, & \text{if } z' = y_s, \\ y_{j+1}, & \text{if } z' = y_j,\ 1 \leq j < s. \end{cases}$$

We define a parity automaton $\mathcal{P}_{\mathcal{ACD}(\mathcal{A})} = (P, \Sigma, P_0, \Delta_P, \Gamma_P, \beta)$ (*ACD transform of* $\mathcal{A}$) equivalent to $\mathcal{A}$ as follows:

States. The states of $\mathcal{P}_{\mathcal{ACD}(\mathcal{A})}$ are of the form (q, x), for $q \in Q$ and x a leaf of the tree associated to q. Initial states are of the form (q_0, x) with $q_0 \in Q_0$ is an initial state in $\mathcal{A}$ and x is the leftmost leaf on its corresponding tree.

$$P = \bigcup_{q \in Q} \{q\} \times \mathit{Leaves}(\mathcal{T}_q),\ P_0 = \{(q_0, x) : q_0 \in Q_0,\ x \text{ the leftmost leaf in } \mathcal{T}_{q_0}\}.$$

Transitions. For each transition $e = q \xrightarrow{a:C} q'$ in Δ and each state $(q, x) \in P$, let us define a transition $(q, x) \xrightarrow{a:p} (q', y)$ in Δ_P as follows: first, q' is the destination state for the original transition. If q and q' are not in the same SCC then y is defined as the leftmost leaf in $\mathcal{T}_{q'}$ and $p = 1$ (except if all $\mathcal{T}_i$ have height 1 and a rounded root: in that case $p = 0$). Otherwise, if both q and q' belong to the i-th SCC of $\mathcal{A}$, then the destination leaf y is the leftmost descendant of $\mathit{NextBranch}(x, e)$ in $\mathcal{T}_{q'}$.
We define the color p of the transition as $\mathit{Depth}(\mathit{Support}(x, e))$, if the root of $\mathcal{T}_i$ is a round node ($\eta_i(\varepsilon) \vDash \alpha$), or as $\mathit{Depth}(\mathit{Support}(x, e)) + 1$ otherwise. We remark that in this way, p is even if and only if $\eta_i(z) \vDash \alpha$.

Parity condition. The condition β is a parity min even condition (cf. Table 1).

Remark 2. If the color 0 does not appear on any transition then we shift all colors by -1 and replace β by a *parity min odd* condition.

Proposition 1 ([6]). *The automaton $\mathcal{P}_{\mathcal{ACD}(\mathcal{A})}$ recognizes $\mathcal{L}(\mathcal{A})$.*

Remark 3. The ACD transformation preserves many properties (determinism, completeness, good-for-gameness, unambiguity...) of the automaton $\mathcal{A}$, see [6].

Remark 4. Since the number of colors used by $\mathcal{P}_{\mathcal{ACD}(\mathcal{A})}$ is at most the height of a tree in $\mathcal{ACD}(\mathcal{A})$, we obtain that $\mathcal{P}_{\mathcal{ACD}(\mathcal{A})}$ never uses more colors than $|\Gamma|+1$. Furthermore, since the TELA does not require all transitions to have a color, we can omit the maximal one and produce an automaton with at most $|\Gamma|$ colors.

In order to state the optimality of this transformation we introduce the notion of *locally bijective morphisms* of automata. Given an automaton $\mathcal{A} = (Q, \Sigma, Q_0, \Delta, \Gamma, \alpha)$ and $q \in Q$, we denote $Out_{\mathcal{A}}(q)$ the set of outgoing transitions of q, i.e., $Out_{\mathcal{A}}(q) = \{q \xrightarrow{a:C} q' \in \Delta \;:\; a \in \Sigma, C \subseteq \Gamma, q' \in Q\}$.

Definition 3 ([6]). *Let $\mathcal{A} = (Q, \Sigma, Q_0, \Delta, \Gamma, \alpha)$ and $\mathcal{A}' = (Q', \Sigma, Q'_0, \Delta', \Gamma', \alpha')$ be two EL automata over Σ. A* locally bijective morphism *from $\mathcal{A}$ to $\mathcal{A}'$ (denoted $\varphi\colon \mathcal{A} \to \mathcal{A}'$) is a pair of maps $\varphi_Q\colon Q \to Q'$, $\varphi_\Delta : \Delta \to \Delta'$ such that:*

- *$\varphi_Q|_{Q_0}$ is a bijection between Q_0 and Q'_0.*
- *$\varphi_\Delta\big(q_1 \xrightarrow{a:C} q_2\big) = \varphi_Q(q_1) \xrightarrow{a:C'} \varphi_Q(q_2)$ for some $C' \subseteq \Gamma'$.*
- *For every $q \in Q$, $\varphi_\Delta|_{Out_{\mathcal{A}}(q)}$ is a bijection between $Out_{\mathcal{A}}(q)$ and $Out_{\mathcal{A}'}(\varphi_Q(q))$*
- *For every run $\varrho \in \Delta^\omega$ in $\mathcal{A}$, ϱ is accepting iff $\varphi_\Delta(\varrho)$ is accepting in $\mathcal{A}'$.*

Theorem 1 ([6]). *Let $\mathcal{A}$ be an Emerson-Lei automaton, and let $\mathcal{P}_{\mathcal{ACD}(\mathcal{A})}$ be the parity automaton obtained by applying the ACD transformation. Then,*

- *There is a locally bijective morphism $\varphi : \mathcal{P}_{\mathcal{ACD}(\mathcal{A})} \to \mathcal{A}$.*
- *If $\mathcal{P}'$ is a parity automaton admitting a locally bijective morphism to $\mathcal{A}$, then $|\mathcal{P}_{\mathcal{ACD}(\mathcal{A})}| \leq |\mathcal{P}'|$.*
- *If $\mathcal{P}'$ is a parity automaton recognizing $\mathcal{L}(\mathcal{A})$, $\mathcal{P}'$ uses at least as many colors as $\mathcal{P}_{\mathcal{ACD}(\mathcal{A})}$.*

Note that all state-duplicating constructions mentioned in the introduction create locally bijective morphisms. Thus the above theorem shows that the ACD transformation duplicates the least number of states.

5.2 Experimental Results

Figures 1 and 2 compare four different paritization procedures applied to 1065 TELA generated[5] from LTL formulas from the Synthesis Competition. These automata have between 2 and 55 colors (mean 5.92, median 5) and between 1 and 245761 states (mean 2023.20, median 20). Automata with fewer than 2 colors have been ignored since they are trivial to paritize.

The procedures are Owl's and Spot's implementation of ACD transform, as well as Spot's implementation of the Zielonka Tree transform [6], and Spot's previous paritization function (called `to_parity`) [28]. We refer the reader to Section 8 for information about the used versions. Two dotted lines on the sides

[5] We used `ltl2tgba -G -D` from Spot, and `ltl2dela` from Owl.

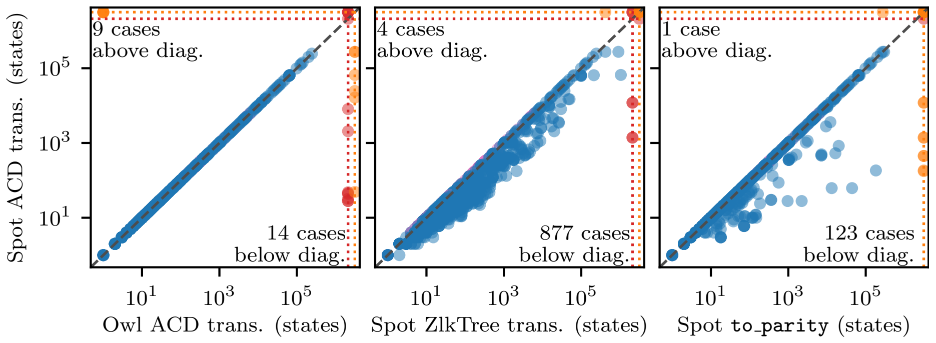

Fig. 1: Comparison of the output size of the four paritization procedures.

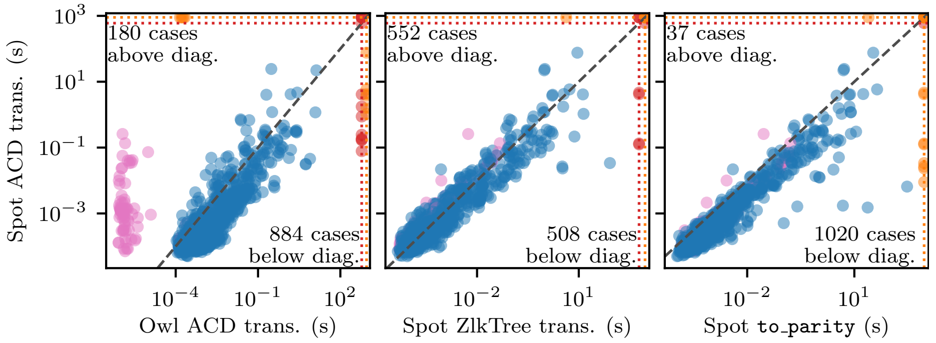

Fig. 2: Time spent performing these four paritization procedures.

of the plots hold cases that did not finish within 500 seconds (red, inner line), or where the tool reported an error[6] (orange, outer line). Pink dots represent input automata that already have parity acceptance: for those, running the ACD transform still makes sense as it will produce an output with a minimal number of colors. However, Owl's implementation, which mostly cares about reducing the number of states, uses a shortcut and will return the input automaton unmodified in this case: this explains the pink cloud on the left of Figure 2.

Owl's and Spot's implementations of the ACD transform produce automata with the same size, as expected. The cases that are not on the diagonal all correspond to timeouts or tool errors. The Zielonka Tree transform, which does not take the automaton structure into consideration, produces automata that are on the average 2.11 times bigger (median 1.60), while its runtime is on the average 6.55 times slower (median 0.97). Lastly, Spot's `to_parity` function is not far from the optimal size given by ACD transform: on the average its output is 3.28 times larger, but the median of that size ratio is 1.00. Similarly, it is on the average 15.94 times slower, but with a median of 1.04.

[6] Either "out-of-memory", or "too many colors" as Spot is restricted to 32 colors.

5.3 ACD Transformation Towards State-Based Parity Automata

Sometimes it is desired to obtain an automaton with the acceptance defined over states. A *state-based parity automaton* is a tuple $\mathcal{A} = (Q, \Sigma, Q_0, \Delta, \phi\colon Q \to \mathbb{N})$ where (Q, Σ, Q_0, Δ) is the underlying structure defined as for transition-based automata in Section 2 (with the only difference that $\Delta \subseteq Q \times \Sigma \times Q$ now), and $\phi\colon Q \to \mathbb{N}$ is a map associating colors to states. A run over $\mathcal{A}$ is *accepting* if the minimal color visited infinitely often is even.

Let $\mathcal{A}$ be a TELA with $\mathcal{ACD}(\mathcal{A}) = \langle \mathcal{T}_1, \dots, \mathcal{T}_k \rangle$. We define an equivalent state-based parity automaton $\mathcal{P}_{sb\text{-}\mathcal{ACD}(\mathcal{A})} = (P, \Sigma, P_0, \Delta_P, \phi\colon P \to \mathbb{N})$ as follows:

States. States are of the form (q, x), for $q \in Q$ and $x \in T_q$ (now the second component corresponds to a node of the ACD that is not necessarily a leaf). The set of initial states is the same as for $\mathcal{P}_{\mathcal{ACD}(\mathcal{A})}$:

$$P = \bigcup_{q \in Q} \{q\} \times T_q\,, \quad P_0 = \{(q_0, x) \,:\, q_0 \in Q_0,\, x \text{ the leftmost leaf in } \mathcal{T}_{q_0}\}.$$

Transitions. For each transition $e = q \xrightarrow{a:C} q' \in \Delta$ and $(q, x) \in P$ we define one transition $(q, x) \xrightarrow{a} (q', y) \in \Delta_P$. To specify the destination node y, we distinguish two cases:

Suppose that x is a leaf in $\mathcal{T}_q$. If $\mathit{NextBranch}(x, e)$ is not the leftmost child of $\mathit{Support}(x, e)$ in $\mathcal{T}_{q'}$, then y is the leftmost leaf below $\mathit{NextBranch}(x, e)$ in $\mathcal{T}_{q'}$ (as in the transition-based case). If $\mathit{NextBranch}(x, e)$ is the leftmost child (a "lap" around $\mathit{Support}(x, e)$ is finished), then we set $y = \mathit{Support}(x, e)$.

If x is not a leaf in $\mathcal{T}_q$, the destination y is determined exactly as if the transition started in (q, x') for x' the leftmost leaf in T_q under x.

Parity condition. $\phi((q, x)) = \mathit{Depth}(x)$, if the root of $\mathcal{T}_q$ is a round node, and $\phi((q, x)) = \mathit{Depth}(x) + 1$ otherwise.

Note that we do not have the same optimality guarantee as in the transition-based case: If x is not a leaf in its corresponding tree, then the states of the form $(q, x) \in P$ are not necessarily reachable in $\mathcal{P}_{sb\text{-}\mathcal{ACD}(\mathcal{A})}$. We only need to add those that can be reached from the initial state. However, the set of reachable states does depend on the ordering of the children in the trees of the ACD, and therefore the size of the final automaton depends on this ordering.

We propose a heuristic to order the children of nodes in $\mathcal{ACD}(\mathcal{A})$. Let $\mathcal{T}_i$ be a tree in $\mathcal{ACD}(\mathcal{A})$ and $x \in T_i$. We define:

$$D_i(x) = \{q' \in Q \,:\, q \xrightarrow{a} q' \notin \eta_i(x), \text{ for some } q \in \mathit{States}(\eta_i(x)), a \in \Sigma\}.$$

The heuristic consists in ordering the children of a node $\mathcal{T}_i$ by decreasing $|D_i(x)|$. Experiments involving transformations towards state-based automata and testing this heuristic can be found in Section 6.2.

6 Degeneralization of Generalized Büchi Automata

The transformation of generalized-Büchi automata with n colors into Büchi automata (with a single color) is known as "*degeneralization*" and has been a very common processing step between algorithms that translate temporal-logic formulas into generalized-Büchi automata, and model-checking algorithms that (used to) only work with Büchi automata. While it initially consisted in making 2^n copies of the GBA [30, Appendix B] to remember the set of colors that had yet to be seen, degeneralization to state-based Büchi acceptance can be done using only $n+1$ copies once an arbitrary order of colors has been selected [13]. A similar construction to transition-based Büchi acceptance requires only n copies of the original automaton. Different orders of colors may lead to a different numbers of reachable states in the Büchi automaton. Some tools even attempted to start the degeneralization in different copies to reduce the number of reachable states [14]. Nowadays, an implementation such as the degeneralization of Spot implements several SCC-based optimizations [2] to reduce the number of output states, but is still sensitive to the arbitrary order selected for colors.

6.1 Transition-based Degeneralization

This order-sensitivity of the degeneralization, even in its transition-based variant, makes a striking difference with ACD. When applied to a generalized Büchi automaton that has some accepting and rejecting paths, the ACD-transform produces an automaton with acceptance Inf(⓿)∨Fin(❶). Since all transitions are either labeled by ⓿ or ❶, color ❶ is superfluous[7] and the condition can be reduced to Inf(⓿). In this context, ACD-transform therefore gives us a transition-based Büchi automaton by duplicating the fewest number of states (Theorem 1(2)).

It can be seen that the cycling around the different children of the ACD (whose ordering is arbitrary) performed during ACD-transform is similar to the process used in traditional degeneralization. What makes the latter sensitive to color ordering is that it only "sees" one transition at a time, while the ACD provides a view of the cycles. For instance a degeneralization would process the sequence $(x) \xrightarrow{⓿} (y) \xrightarrow{❶} (z)$ differently from the sequence $(x) \xrightarrow{❶} (y) \xrightarrow{⓿} (z)$ depending on the order in which colors are expected to be encountered. However, if there is no other transition reaching or leaving (y) the two colors will always be seen together so their order should not matter: the two transitions belong to the same node of the ACD. The *propagation of colors* [28] is a related preprocessing step that can improve the degeneralization by propagating all colors common to the incoming transitions of a state to its outgoing transitions and vice-versa. It would turn the previous situation into $(x) \xrightarrow{⓿❶} (y) \xrightarrow{⓿❶} (z)$ making the color order selected by the degeneralization irrelevant (in this case).

A comparison of the output size of the traditional degeneralization implemented in Spot (which includes several optimizations learned over the years)

[7] In an automaton with "parity min" acceptance where all transitions are colored, the maximal color can always be omitted and replaced by the empty set.

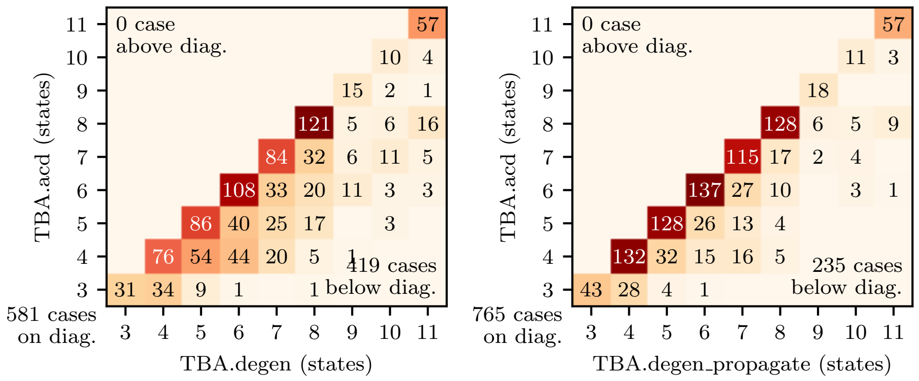

Fig. 3: Two-dimensional histogram of the sizes of 1000 automata, degeneralized to transition-based Büchi automata, using Spot's degeneralization function (with or without propagation of colors), or using ACD-transform.

against that of ACD-transform is given in the left plot of Figure 3. Unsurprisingly, because of ACD-transform's optimality, there are no cases where ACD loses to Spot's transition-based degeneralization. The use of the *propagation of colors* (right of the plot) is an improvement (the non-optimal cases dropped from 419 to 235) but not a cure.

Remark 5. The input automata used in this section and the next one is a set of 1000 randomly generated, minimal, deterministic, transition-based generalized Büchi automata, with 3 or 4 states and 2 or 3 colors. The reason for using such small minimal automata is to be able to use a SAT-based minimization [1] on the degeneralized state-based output in the next section to estimate how large the gap between an optimal and our procedure is.

6.2 State-based degeneralization

If ACD is used to produce a state-based output, as explained in Subsection 5.3, the obtained automaton is not guaranteed to be minimal with respect to locally bijective morphisms. In this case we can obtain a weaker optimality result:

Proposition 2. *Let $\mathcal{A}$ be a generalized Büchi automaton, and let $\mathcal{B}_{sb-\mathcal{ACD}(\mathcal{A})}$ be the state-based Büchi automaton obtained by applying the ACD state-based transformation. If $\mathcal{B}'$ be is a state-based Büchi automaton admitting a locally bijective morphism to $\mathcal{A}$, then $|\mathcal{B}_{sb-\mathcal{ACD}(\mathcal{A})}| \leq |\mathcal{B}'| + |\mathcal{A}|$.*

Proof. Let $\mathcal{B}'$ be a state-based Büchi automaton admitting a locally bijective morphism to $\mathcal{A}$. We can transform it into a transition-based Büchi automaton $\mathcal{B}'_{trans}$ by setting the transitions leaving accepting states to be accepting. This automaton has the same size than $\mathcal{B}'$ and it also accepts a locally bijective morphism to $\mathcal{A}$. Therefore, by Theorem 1, we have that $|\mathcal{B}_{\mathcal{ACD}(\mathcal{A})}| \leq |\mathcal{B}'_{trans}| =$

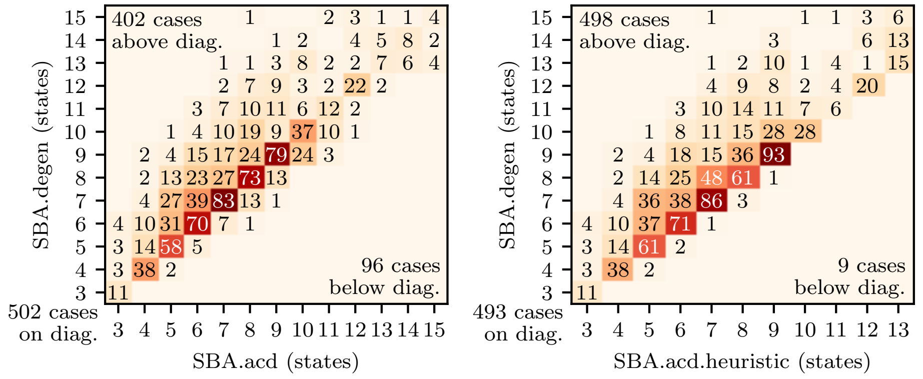

Fig. 4: Comparison of three ways to degeneralize to state-based Büchi: (acd, acd.heuristic) using the state-based version of ACD-transform with or without heuristic, and (degen) classical degeneralization.

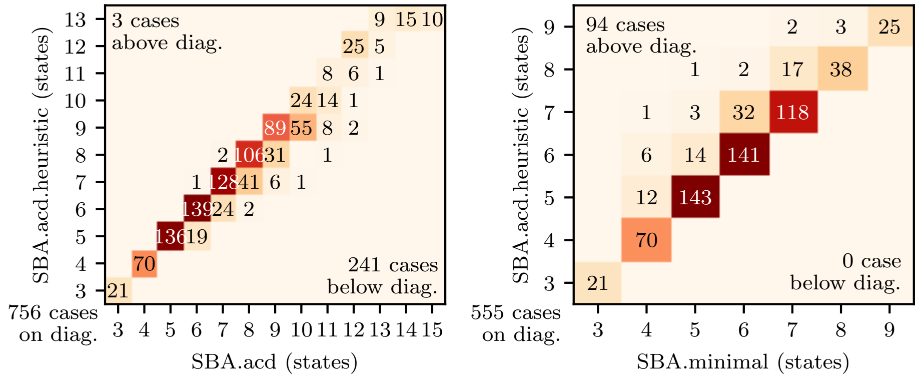

Fig. 5: Effect of the heuristic for ordering children of the ACD, and comparison to the minimal degeneralized automata (when known).

$|\mathcal{B}'|$, where $\mathcal{B}_{\mathcal{ACD}(\mathcal{A})}$ is the transition-based automaton obtained applying the ACD-transformation. We claim that $|\mathcal{B}_{sb-\mathcal{ACD}(\mathcal{A})}| \leq |\mathcal{B}_{\mathcal{ACD}(\mathcal{A})}| + |\mathcal{A}|$ (therefore implying that $|\mathcal{B}_{sb-\mathcal{ACD}(\mathcal{A})}| \leq |\mathcal{B}'| + |\mathcal{A}|$). Indeed, the set of states of $\mathcal{B}_{sb-\mathcal{ACD}(\mathcal{A})}$ is the union of the set of states of $\mathcal{B}_{\mathcal{ACD}(\mathcal{A})}$ and a subset of nodes of the form (q, ε), where ε is the root of T_q. There are at most $|\mathcal{A}|$ nodes of this form. $\square$

Figure 4 compares three ways to perform state-based degeneralization. The ACD comes in two variants, with or without the heuristic of Section 5.3, and it is compared against the state-based degeneralization of Spot.

Figure 5 shows how the heuristic variant compares to the one without, and how it compares with the size of a minimal DBA, when its size could be computed in reasonable time (in 649 cases). Note that there might not be a local bijective morphism between the input automaton and the minimal DBA computed this way, nonetheless these minimal size automata can serve as a reference point to estimate the quality of a degeneralization. Compared to this subset of minimal DBA, the average number of additional states produced by the state-based ACD is 0.17 with heuristics, and 0.33 without. Comparatively, Spot's degeneralization has an average of 1.21 extra states.

7 Deciding Typeness

We highlight now how the ACD can be used to decide *typeness* of deterministic TELA. This problem, first introduced by Krishnan and Brayton [19], consists of deciding whether we can replace the acceptance condition of a given automaton by another (hopefully simpler) without changing the transition structure and preserving the language (see Table 1 for a list of common acceptance conditions).

Let $\mathcal{A} = (Q, \Sigma, Q_0, \Delta, \Gamma, \alpha)$ be a TELA. We say that $\mathcal{A}$ is X*-type*, for $X \in \{\mathrm{B}, \mathrm{C}, \mathrm{GB}, \mathrm{GC}, \mathrm{P}, \mathrm{R}, \mathrm{S}\}$, if there is an X-automaton over the same structure, $\mathcal{A}' = (Q, \Sigma, Q_0, \Delta', \Gamma', \beta)$ (where Δ and Δ' only differ on the coloring of the transitions), such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$ and β belongs to X. We emphasize that we permit to use a different set of colors Γ' in $\mathcal{A}'$. Some conditions can always be rewritten as conditions of other kinds (for example, Büchi conditions can be expressed as parity ones, so being B-type implies being P-type). We should not confuse this notion with the expressive power of deterministic automata using these conditions. For example, both deterministic parity automata and Rabin automata recognize all ω-regular languages, but there are Rabin automata that are not parity-type. Further, we say that an automaton $\mathcal{A}$ is *weak* if for every SCC $\mathcal{S}$ of $\mathcal{A}$, all cycles in $\mathcal{S}$ are accepting or all of them are rejecting.

The following result shows that the ACD is a sufficient data structure for deciding typeness for many common acceptance conditions. We remark that the second item adds to the results of Casares et al. [7] (this statement only holds if transitions of automata are labeled with subsets of colors, which is not allowed in their model).

Proposition 3 ([7, Section 5.2]). *Let $\mathcal{A}$ be a deterministic TELA such that all its states $q \in Q$ are reachable and let $\mathcal{ACD}(\mathcal{A}) = \langle \mathcal{T}_1, \dots, \mathcal{T}_k \rangle$ be its Alternating Cycle Decomposition. Then the following statements hold:*

1. *$\mathcal{A}$ is Rabin-type (resp. Streett type) if and only if for every $q \in Q$, every round node (resp. square node) of $\mathcal{T}_q$ has at most one child in $\mathcal{T}_q$. It is parity-type if and only if it is both Rabin and Streett-type.*
2. *$\mathcal{A}$ is generalized Büchi-type (resp. generalized co-Büchi-type) if and only if for every $1 \le i \le k$, Height$(\mathcal{T}_i) \le 2$ and in case of equality, the root of T_i is a round node (resp. square node).*
3. *$\mathcal{A}$ is weak if and only if for every $1 \le i \le k$, Height$(\mathcal{T}_i) = 1$.*

Also, the least number of colors used by a deterministic parity automaton recognizing $\mathcal{L}(\mathcal{A})$ is $\max\limits_{1 \le i \le k} \mathit{Height}(T_i) + \nu$, where $\nu = 0$ if the root of all trees of maximal height have the same shape (round or square), and $\nu = 1$ otherwise.

If one of the previous conditions holds, then $\mathcal{ACD}(\mathcal{A})$ also provides an effective procedure to relabel $\mathcal{A}$ with the corresponding acceptance condition.

Remark 6. The ACD gives a typeness result for each SCC of the automaton, which allows to simplify the acceptance condition of each of them independently. Further, implications from right to left in Proposition 3 also hold for non-deterministic automata.

Proposition 3 provides an effective procedure to check typeness of TELA: we just have to build the ACD and verify that it has the appropriate shape. Spot's implementation of ACD has options to abort the construction as soon as it detects that the shape is wrong. Moreover, if an automaton is parity-type, the ACD provides a method to relabel the automaton with a minimal number of colors. Finally, if the automaton already has parity acceptance, the ACD transformation boils down to the algorithm of Carton and Maceiras [5].

8 Availability

The ACD and the transformations based on it are currently implemented in two open-source tools: Spot 2.10 [9] and Owl 21.0 [18]. (The original developments were independent before the authors met and worked on this joint paper.)

In Spot 2.10, the ACD can be played with using the Python bindings. The `acd` class implements the decomposition, and will render it as an interactive forest of nodes that can be clicked to highlight the relevant cycles in the input automaton. The `acd_transform()` and `acd_transform_sbacc()` implements the transition-based and state-based variant of the paritization procedure. Additionally, the `acd` class has options to heuristically order the children to favor the state-based construction, or to abort the construction as soon as it is clear that the ACD does not have Rabin or Street shape (in case one wants to use it to establish typeness of automata). All these features are illustrated at https://spot.lrde.epita.fr/ipynb/zlktree.html. In the future, ACD will be used more by the rest of Spot, and will be one option of the `ltlsynt` tool (for LTL synthesis).

In Owl, the ACD transformation is available through the `aut2parity` command. This command reads an automaton in the HOA format [3] using arbitrary acceptance, and produces a parity automaton in the same format. The tool Strix [23], which builds upon Owl, gained in version 21.0.0 the option to use the ACD-construction as an intermediate step.

Instructions to reproduce all experiments and included in the artifact [8].

9 Conclusion

We have shown that ACD is more than a theoretically-appealing construction: our two implementations show that the construction is very usable in practice, and provide a baseline for further improvements. We have also shown that ACD is a Swiss-army knife for ω-automata in the sense that it can generalize and replace several specific constructions (paritization, degeneralization, typeness checks).

References

1. Baarir, S., Duret-Lutz, A.: Mechanizing the minimization of deterministic generalized Büchi automata. In: Proceedings of the 34th IFIP International Conference on Formal Techniques for Distributed Objects, Components and Systems (FORTE'14), Lecture Notes in Computer Science, vol. 8461, pp. 266–283, Springer (Jun 2014), https://doi.org/10.1007/978-3-662-43613-4_17
2. Babiak, T., Badie, T., Duret-Lutz, A., Křetínský, M., Strejček, J.: Compositional approach to suspension and other improvements to LTL translation. In: Proceedings of the 20th International SPIN Symposium on Model Checking of Software (SPIN'13), Lecture Notes in Computer Science, vol. 7976, pp. 81–98, Springer (Jul 2013), https://doi.org/10.1007/978-3-642-39176-7_6
3. Babiak, T., Blahoudek, F., Duret-Lutz, A., Klein, J., Křetínský, J., Müller, D., Parker, D., Strejček, J.: The hanoi omega-automata format. In: Kroening, D., Păsăreanu, C.S. (eds.) Computer Aided Verification, pp. 479–486, Springer International Publishing (2015)
4. Battiti, R., , Protasi, M.: Handbook of Combinatorial Optimization: Volume 1–3, chap. Approximate Algorithms and Heuristics for MAX-SAT, pp. 77–148. Springer US (1998), ISBN 978-1-4613-0303-9, https://doi.org/10.1007/978-1-4613-0303-9_2
5. Carton, O., Maceiras, R.: Computing the Rabin index of a parity automaton. Informatique théorique et applications **33**(6), 495–505 (1999), URL http://www.numdam.org/item/ITA_1999__33_6_495_0/
6. Casares, A., Colcombet, T., Fijalkow, N.: Optimal transformations of games and automata using Muller conditions. In: Bansal, N., Merelli, E., Worrell, J. (eds.) Proceedings of the 48th International Colloquium on Automata, Languages, and Programming (ICALP'21), Leibniz International Proceedings in Informatics (LIPIcs), vol. 198, pp. 123:1–123:14, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany (2021), https://doi.org/10.4230/LIPIcs.ICALP.2021.123
7. Casares, A., Colcombet, T., Fijalkow, N.: Optimal transformations of muller conditions. Extended version of [6], on ArXiv. (2021), https://arxiv.org/abs/2011.13041
8. Casares, A., Duret-Lutz, A., Meyer, K.J., Renkin, F., Sickert, S.: Artifact for the paper "Practical applications of the alternating cycle decomposition". https://doi.org/10.5281/zenodo.5572613 (2021)
9. Duret-Lutz, A., Lewkowicz, A., Fauchille, A., Michaud, T., Renault, E., Xu, L.: Spot 2.0 — a framework for LTL and ω-automata manipulation. In: Proceedings of the 14th International Symposium on Automated Technology for Verification and Analysis (ATVA'16), Lecture Notes in Computer Science, vol. 9938, pp. 122–129, Springer (Oct 2016), https://doi.org/10.1007/978-3-319-46520-3_8
10. Emerson, E.A., Lei, C.L.: Modalities for model checking (extended abstract): Branching time strikes back. In: Proceedings of the 12th ACM symposium on Principles of Programming Languages (POPL'85), pp. 84–96, ACM (1985), https://doi.org/10.1145/318593.318620
11. Esparza, J., Křetínský, J., Raskin, J.F., Sickert, S.: From LTL and limit-deterministic Büchi automata to deterministic parity automata. In: Proceedings of the 23rd International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'17), Lecture Notes in Computer Science, vol. 10205, pp. 426–442, Springer-Verlag (2017), https://doi.org/10.1007/978-3-662-54577-5_25
12. Esparza, J., Křetínský, J., Sickert, S.: A unified translation of linear temporal logic to ω-automata. J. ACM **67**(6) (Oct 2020), https://doi.org/10.1145/3417995

13. Gastin, P., Oddoux, D.: Fast LTL to Büchi automata translation. In: Berry, G., Comon, H., Finkel, A. (eds.) Proceedings of the 13th International Conference on Computer Aided Verification (CAV'01), Lecture Notes in Computer Science, vol. 2102, pp. 53–65, Springer-Verlag (2001), https://doi.org/10.1007/3-540-44585-4_6
14. Giannakopoulou, D., Lerda, F.: From states to transitions: Improving translation of LTL formulæ to Büchi automata. In: Peled, D., Vardi, M. (eds.) Proceedings of the 22nd IFIP WG 6.1 International Conference on Formal Techniques for Networked and Distributed Systems (FORTE'02), Lecture Notes in Computer Science, vol. 2529, pp. 308–326, Springer-Verlag, Houston, Texas (Nov 2002)
15. Grädel, E., Thomas, W., Wilke, T. (eds.): Automata Logics, and Infinite Games. Springer, Berlin, Heidelberg (2002), https://doi.org/10.1007/3-540-36387-4
16. Gurevich, Y., Harrington, L.: Trees, automata, and games. In: Proceedings of the 14th annual ACM symposium on Theory of computing (STOC'82), pp. 60–65 (1982), https://doi.org/10.1145/800070.802177
17. Jacobs, S., Bloem, R., Colange, M., Faymonville, P., Finkbeiner, B., Khalimov, A., Klein, F., Luttenberger, M., Meyer, P.J., Michaud, T., Sakr, M., Sickert, S., Tentrup, L., Walker, A.: The 5th reactive synthesis competition (SYNTCOMP 2018): Benchmarks, participants & results. CoRR **abs/1904.07736** (2019), URL http://arxiv.org/abs/1904.07736
18. Kretínský, J., Meggendorfer, T., Sickert, S.: Owl: A library for ω-words, automata, and LTL. In: Proceedings of the 16th International Symposium on Automated Technology for Verification and Analysis (ATVA'18), Lecture Notes in Computer Science, vol. 11138, pp. 543–550, Springer (2018), https://doi.org/10.1007/978-3-030-01090-4_34
19. Krishnan, Sriram C.and Puri, A., Brayton, R.K.: Deterministic ω automata vis-a-vis deterministic buchi automata. In: Algorithms and Computation, pp. 378–386, Springer Berlin Heidelberg, Berlin, Heidelberg (1994)
20. Křetínský, J., Meggendorfer, T., Waldmann, C., Weininger, M.: Index appearance record for transforming Rabin automata into parity automata. In: Legay, A., Margaria, T. (eds.) Proceedings of the 23st International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'17), Lecture Notes in Computer Science, vol. 10205, pp. 443–460 (2017), https://doi.org/10.1007/978-3-662-54577-5_26
21. Křetínský, J., Meggendorfer, T., Waldmann, C., Weininger, M.: Index appearance record with preorders. Acta Informatica (2021), https://doi.org/10.1007/s00236-021-00412-y
22. Löding, C.: Optimal bounds for transformations of ω-automata. In: Proceedings of the 19th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'99), Lecture Notes in Computer Science, vol. 1738, pp. 97–109, Springer (1999), https://doi.org/10.1007/3-540-46691-6_8
23. Luttenberger, M., Meyer, P.J., Sickert, S.: Practical synthesis of reactive systems from LTL specifications via parity games. Acta Informatica pp. 3–36 (2020), https://doi.org/10.1007/s00236-019-00349-3
24. Löding, C.: Methods for the Transformation of ω-Automata: Complexity and Connection to Second Order Logic. Master's thesis, Institute of Computer Science and Applied Mathematics Christian-Albrechts-University of Kiel (1998), URL https://old.automata.rwth-aachen.de/users/loeding/diploma_loeding.pdf
25. Meyer, P., Sickert, S.: On the optimal and practical conversion of Emerson-Lei automata into parity automata. Unpublished manuscript, obsoleted by the work of Casares et al. [6]. (2021)

26. Michaud, T., Colange, M.: Reactive synthesis from LTL specification with Spot. In: Proceedings of the 7th Workshop on Synthesis, SYNT@CAV 2018, Electronic Proceedings in Theoretical Computer Science (2018)
27. Pnueli, A., Rosner, R.: On the synthesis of a reactive module. In: Proceedings of the 16th ACM SIGPLAN-SIGACT symposium on Principles of programming languages (POPL'89), pp. 179—190 (1989), https://doi.org/10.1145/75277.75293
28. Renkin, F., Duret-Lutz, A., Pommellet, A.: Practical "paritizing" of Emerson-Lei automata. In: Proceedings of the 18th International Symposium on Automated Technology for Verification and Analysis (ATVA'20), Lecture Notes in Computer Science, vol. 12302, pp. 127–143, Springer (Oct 2020), https://doi.org/10.1007/978-3-030-59152-6_7
29. Vardi, M.Y.: An automata-theoretic approach to linear temporal logic. In: Logics for Concurrency: Structure versus Automata, volume 1043 of Lecture Notes in Computer Science, pp. 238–266, Springer-Verlag (1996)
30. Vardi, M.Y., Wolper, P.: An automata-theoretic approach to automatic program verification. In: Proceedings of the 1st Symposium on Logic in Computer Science (LICS'86), pp. 332–344, IEEE Computer Society Press (Jun 1986)
31. Zielonka, W.: Infinite games on finitely coloured graphs with applications to automata on infinite trees. Theoretical Computer Science **200**(1), 135–183 (1998), https://doi.org/10.1016/S0304-3975(98)00009-7