The Complexity of Simplifying ω-Automata Through the Alternating Cycle Decomposition

Antonio Casares
University of Warsaw, Poland

Corto Mascle
LaBRI, Université de Bordeaux, France

Abstract

In 2021, Casares, Colcombet and Fijalkow introduced the Alternating Cycle Decomposition (ACD), a structure used to define optimal transformations of Muller into parity automata and to obtain theoretical results about the possibility of relabelling automata with different acceptance conditions. In this work, we study the complexity of computing the ACD and its DAG-version, proving that this can be done in polynomial time for suitable representations of the acceptance condition of the Muller automaton. As corollaries, we obtain that we can decide typeness of Muller automata in polynomial time, as well as the parity index of the languages they recognise.

Furthermore, we show that we can minimise in polynomial time the number of colours (resp. Rabin pairs) defining a Muller (resp. Rabin) acceptance condition, but that these problems become NP-complete when taking into account the structure of an automaton using such a condition.

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

Keywords and phrases Omega-regular languages, Muller automata, Zielonka tree

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.35

Related Version *Full Version*: https://arxiv.org/abs/2401.03811 [13]

Funding Antonio Casares: Supported by the Polish National Science Centre (NCN) grant "Polynomial finite state computation" (2022/46/A/ST6/00072).

This document contains hyperlinks. Each occurrence of a notion is linked to its *definition*. On an electronic device, the reader can click on words or symbols (or just hover over them on some PDF readers) to see their definition.

1 Introduction

1.1 Context

Automata for the synthesis problem. Since the 60s, automata over infinite words have provided a fundamental tool to study problems related to the decidability of different logics [5, 38]. Recent focus has centered on the study of synthesis of controllers for reactive systems with the specification given in Linear Temporal Logic (LTL). The original automata-theoretic approach by Pnueli and Rosner [37] remains at the heart of the state-of-the-art LTL-synthesis tools [19, 29, 33, 35]. Their method consists in translating the LTL formula into a deterministic ω-automaton which is then used to build an infinite duration game; a winning strategy in this game provides a correct controller for the system.

Different acceptance conditions. There are different ways of specifying which runs of an automaton over infinite words are accepting. Generally, we label the transitions of the automaton with some output colours, and we then indicate which colours should be seen (or not) infinitely often. This can be expressed in a variety of ways, obtaining different acceptance conditions, such as parity, Rabin or Muller. The complexity of such acceptance conditions is crucial in the performance of algorithms dealing with automata and games over infinite words. For instance, parity games can be solved in quasi-polynomial time [6] and parity games solvers are extremely performing in practice [24], while solving Rabin and Muller games is, respectively, NP-complete [18] and PSPACE-complete [22]. Moreover, many existing algorithms for solving these games are polynomial on the size of the game graph, and are exponential only on parameters from the acceptance condition: Muller games can be solved in time $\mathcal{O}(k^{5k}n^5)$ [6, Theorem 3.4], where n is the size of the game and k is the number of colours used by the acceptance condition, and Rabin games can be solved in time $\mathcal{O}(n^{r+3}rr!)$ [36, Theorem 7], where r is the number of Rabin pairs of the acceptance condition. Also, the emptiness check of Muller automata with the condition represented by a Boolean formula ϕ (Emerson-Lei condition) can be done in time $\mathcal{O}(2^k kn^2|\phi|)$ [2, Theorem 1].

Some important objectives are therefore: (1) transform an automaton $\mathcal{A}$ using a complex acceptance conditions into an automaton $\mathcal{B}$ using a simpler one, and (2) simplify as much as possible the acceptance condition used by an automaton $\mathcal{A}$ (without adding further states).

The Zielonka tree and Zielonka DAG. The Zielonka tree is an informative representation of Muller conditions, introduced for the study of strategy complexity in Muller games [42, 17]. Zielonka showed that we can use this structure to tell whether a Muller language can be expressed as a Rabin or a parity language [42, Section 5]. Moreover, it has been recently proved that the Zielonka tree provides minimal deterministic parity automata recognising a Muller condition [10, 31], and can thus be used to transform Muller automata using this condition into equivalent parity automata.

A natural alternative is to consider the more succinct DAG-version of this structure: the Zielonka DAG. Hunter and Dawar studied the complexity of building the Zielonka DAG from an explicit representation of a Muller condition, and the complexity of solving Muller games for these different representations [23]. Recently, Hugenroth showed that many decision problems concerning Muller automata become tractable when using the Zielonka DAG to represent the acceptance condition [21].

The ACD: Theoretical applications. In 2021, Casares, Colcombet and Fijalkow [9] proposed the Alternating Cycle Decomposition (ACD) as a generalisation of the Zielonka tree. The main motivation for the introduction of the ACD was to define optimal transformations of automata: given a Muller automaton $\mathcal{A}$, we can build using the ACD an equivalent parity automaton that is minimal amongst all parity automata obtained by duplicating states of $\mathcal{A}$ [10, Theorem 5.32]. Moreover, the ACD can be used to tell whether a Muller automaton can be relabelled with an acceptance condition of a simpler type [10, Section 6.1].

However, the works introducing the ACD [9, 10] are of theoretical nature, and no study of the cost of constructing it and performing the related transformations is presented.

The ACD: Practice. The transformations based on the ACD have been implemented in the tools Spot 2.10 [16] and Owl 21.0 [27], and are used in the LTL-synthesis tools `ltlsynt` [33] and `STRIX` [29, 32] (top-ranked in the SYNTCOMP competitions [24]). In the tool paper [12], these transformation are compared with the state-of-the-art methods to transform Emerson-Lei automata into parity ones. Surprisingly, the transformation based on the ACD does not only produce the smallest parity automata, but also outperforms all other existing paritizing methods in computation time.

In [12, Section 4], an algorithm computing the ACD is proposed. However, the focus is made in the handling of Boolean formulas to enhance the algorithm's performance in practice, but no theoretical analysis of its complexity is provided.

Simplification of acceptance conditions. As already mentioned, the complexity of the acceptance conditions play a crucial role in algorithms. One can simplify the acceptance condition of a Muller automaton by adding further states (and the optimal way of doing this is determined by the ACD [10]). However, in some cases this leads to an exponential blow-up in the number of states [28]. A natural question is therefore to try to simplify the acceptance condition while avoiding adding so many states. We consider two versions of this problem:

Typeness problem. Can we relabel the acceptance condition of a Muller automaton with one of a simpler type, such as Rabin, Streett or parity?

Minimisation of colours and Rabin pairs. Can we minimise the number of colours used by the acceptance condition (or, in the case of Rabin automata, the number of Rabin pairs)?

The ACD has proven fruitful for studying the typeness problem: just by inspecting the ACD of $\mathcal{A}$, we can tell whether we can relabel it with an equivalent Rabin, parity or Streett acceptance condition [10]. Also, it is a classical result that we can minimise in polynomial time the number of colours used by a parity automaton [7]. However, it was still unclear whether the ACD could help to minimise the number of colours of Muller conditions or the number of Rabin pairs of Rabin acceptance conditions, question that we tackle in this work.

The minimisation of colours in Muller automata has recently been studied by Schwarzová, Strejček and Major [39]. In their approach, they use heuristics to reduce the number of colours by applying QBF-solvers. The final acceptance condition is however not guaranteed to have a minimal number of colours. There have also been attempts to minimise the number of Rabin pairs of Rabin automata coming from the determinisation of Büchi automata [40]. Also, in their work about minimal history-deterministic Rabin automata, Casares, Colcombet and Lehtinen left open the question of the minimisation of Rabin pairs [11].

1.2 Contributions

1. **Computation of the ACD and the ACD-DAG.** We show that we can compute the ACD of a Muller automaton in polynomial time, provided that the Zielonka tree of its acceptance condition is given as input (Theorem 13). This shows that the computation of the ACD is not harder than that of the Zielonka tree, (partially) explaining the strikingly favourable experimental results from [12]. We also show that we can compute the DAG-version of the ACD in polynomial time if the acceptance condition of $\mathcal{A}$ is given colour-explicitly or by a Zielonka DAG (Theorem 15). The main technical challenge is to prove that the ACD has polynomial size in the size of the Zielonka tree.
2. **Deciding typeness in polynomial time.** Combining the previous contributions with the results from [10], we directly obtain that we can decide in polynomial time whether a Muller automaton can be relabelled with an equivalent parity, Rabin or Streett acceptance condition (Corollary 16). Moreover, we recover a result from Wilke and Yoo [41]: we can compute in polynomial time the parity index of the language of a Muller automaton.
3. **Minimisation of colours and Rabin pairs of acceptance conditions.** For a given Muller (resp. Rabin) language L, we show that we can minimise the number of colours (resp. Rabin pairs) needed to define L in polynomial time (Theorems 20 and 21). We also relate the minimisation of Rabin pairs to a subclass of interest of Boolean formulas, called generalised Horn formulas.
4. **Minimisation of colours and Rabin pairs over an automaton structure.** Given an automaton $\mathcal{A}$ using a Muller (resp. Rabin) acceptance condition, we show that the problem of minimising the number of colours (resp. Rabin pairs) to relabel $\mathcal{A}$ with an equivalent acceptance condition over its structure is NP-complete, and it remains NP-hard even

if the ACD is given as input (Theorems 26 and 27). This came as a surprise to us, as our first intuition was that the ACD would allow to lift the previous polynomial-time minimisation algorithms to ones which take into account the structure of the automaton.

We provide proof ideas for all the results, technical proofs can be found in the full version [13]. The full version also contains further contributions and discussions about the size of different representations of Muller conditions (summarised in Figure 3).

2 Preliminaries

2.1 Automata over infinite words and their acceptance conditions

Given a set Γ, we write 2^Γ_+ for the set of its non-empty subsets. For a word $w \in \Gamma^\omega$, we let $\mathsf{Inf}(w)$ be the set of letters appearing infinitely often in w.

Automata. A *(non-deterministic) automaton* is a tuple $\mathcal{A} = (Q, q_{\mathsf{init}}, \Sigma, \Delta, \Gamma, \mathsf{col}, W)$, where Q is a finite set of states, $q_{\mathsf{init}} \in Q$ is an initial state, Σ is an *input alphabet*, $\Delta \subseteq Q \times \Sigma \times Q$ is a set of transitions, Γ is a finite set of *output colours*, $\mathsf{col}\colon \Delta \to \Gamma$ is a *colouring* of the transitions, and $W \subseteq \Gamma^\omega$ is a language over Γ. We call the tuple (col, W) the *acceptance condition* of $\mathcal{A}$. We write $q \xrightarrow{a} q'$ to denote a transition $e = (q, a, q') \in \Delta$, and $q \xrightarrow{a:c} q'$ to further indicate that $\mathsf{col}(e) = c$. We write $q \overset{w:u}{\rightsquigarrow} q'$ to represent the existence of a path from q to q' labelled with the input letters $w \in \Sigma^*$ and output colours $u \in \Gamma^*$.

Given an automaton $\mathcal{A}$ and a word $w \in \Sigma^\omega$, a *run over* w in $\mathcal{A}$ is a path $q_{\mathsf{init}} \xrightarrow{w_0:c_0} q_1 \xrightarrow{w_1:c_1} q_2 \xrightarrow{w_2:c_2} q_3 \xrightarrow{w_3:c_3} \cdots \in \Delta^\omega$. Such a run is *accepting* if $c_0 c_1 c_2 \cdots \in W$, and *rejecting* otherwise. A word $w \in \Sigma^\omega$ is *accepted* by $\mathcal{A}$ if it admits an accepting run. The *language recognised* by an automaton $\mathcal{A}$ is the set $\mathcal{L}(\mathcal{A}) = \{w \in \Sigma^\omega \mid w \text{ is accepted by } \mathcal{A}\}$. Two automata over the same alphabet are *equivalent* if they recognise the same language. An automaton is *deterministic* (resp. *complete*) if for every $q \in Q$ and $a \in \Sigma$, there is at most (resp. at least) one transition $q \xrightarrow{a} q'$.

We underline that the colours of the acceptance of runs appear *over transitions*. For a discussion on the differences between transition and state-based automata, and arguments in favour of the former, we refer to [8, Chap. VI].

It is sometimes useful to let transitions carry multiple colours – for instance, this is the standard model in the HOA format [1]. For many results of this paper (those from Section 3), allowing or not multiple colours per edge does not make a difference; we could always take 2^Γ or Δ as new set of colours. This will however be relevant in Section 4.2. Also, the HOA format allows for multiple transitions between the same two states with the same input letter. These transitions can always be replaced by one carrying multiple colours (we refer to [11, Prop. 18] for details).

We let the *size of* $\mathcal{A}$ be $|\mathcal{A}| = |Q| + |\Sigma| + |\Delta| + |\Gamma|$. We note that this does not take into account the size of the representation of its acceptance condition, which can admit different forms (see page 7). When necessary, we will indicate the size of the representation of the acceptance condition separately.

Acceptance conditions. We now define the main classes of languages used by automata over infinite words as acceptance conditions. We let Γ stand for a finite set of colours.

Muller. We define the *Muller language* of a family $\mathcal{F} \subseteq 2^\Gamma_+$ of non-empty subsets of Γ as:

$$\mathsf{Muller}_\Gamma(\mathcal{F}) = \{w \in \Gamma^\omega \mid \mathsf{Inf}(w) \in \mathcal{F}\}.$$

We will often refer to sets in $\mathcal{F}$ as *accepting sets* and sets not in $\mathcal{F}$ as *rejecting sets*.

Rabin. A Rabin condition is represented by a family $\mathcal{R} = \{(\mathfrak{g}_1, \mathfrak{r}_1), \dots, (\mathfrak{g}_r, \mathfrak{r}_r)\}$ of *Rabin pairs*, where $\mathfrak{g}_j, \mathfrak{r}_j \subseteq \Gamma$. We define the *Rabin language* of a single Rabin pair $(\mathfrak{g}, \mathfrak{r})$ as

$$\mathsf{Rabin}_\Gamma((\mathfrak{g}, \mathfrak{r})) = \{w \in \Gamma^\omega \mid \mathsf{Inf}(w) \cap \mathfrak{g} \neq \emptyset \wedge \mathsf{Inf}(w) \cap \mathfrak{r} = \emptyset\},$$

and the Rabin language of a family of Rabin pairs $\mathcal{R}$ as: $\mathsf{Rabin}_\Gamma(\mathcal{R}) = \bigcup_{j=1}^{r} \mathsf{Rabin}_\Gamma((\mathfrak{g}_j, \mathfrak{r}_j))$.

Streett. The *Streett language* of a family $\mathcal{R} = \{(\mathfrak{g}_1, \mathfrak{r}_1), \dots, (\mathfrak{g}_r, \mathfrak{r}_r)\}$ of Rabin pairs is defined as the complement of its Rabin language:

$$\mathsf{Streett}_\Gamma(\mathcal{R}) = \Gamma^\omega \setminus \mathsf{Rabin}_\Gamma(\mathcal{R}).$$

Parity. We define the *parity language* over a finite alphabet $\Pi \subseteq \mathbb{N}$ as:

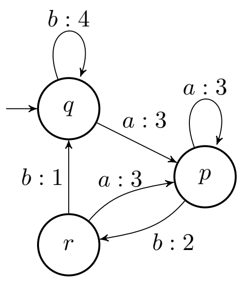
$$\mathsf{parity}_\Pi = \{w \in \Pi^\omega \mid \min \mathsf{Inf}(w) \text{ is even}\}.$$

We say that an automaton is a $\mathcal{C}$ *automaton*, for $\mathcal{C}$ one of the classes of languages above, if its acceptance condition uses a $\mathcal{C}$ language. We refer to the survey [3] for a more detailed account on different types of acceptance conditions.

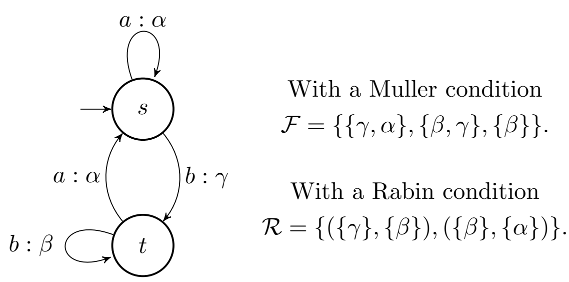
▶ Remark 1. Muller languages are exactly the languages characterised by the set of letters seen infinitely often. They are also the languages recognised by deterministic Muller automata with one state.

We observe that parity languages are special cases of Rabin and Streett languages which are in turn special cases of Muller languages.

▶ **Example 2.** In Figure 1 we show different types of automata over the alphabet $\Sigma = \{a, b\}$ recognising the language of words that contain infinitely many bs and eventually do not encounter the factor abb.



Parity automaton $\mathcal{A}_1$.



Automaton $\mathcal{A}_2$ with equivalent Muller and Rabin conditions over it.

Figure 1 Different types of automata recognising the language $L = \Sigma^* b^\omega + \Sigma^*(a^+b)^\omega$. (Note that the set of outputs that occur infinitely often in a run of $\mathcal{A}_2$ cannot be $\{\beta, \gamma\}$.)

The 8 classes of automata obtained by combining the 4 types of acceptance conditions above with deterministic and non-deterministic models are equally expressive [30, 34]. We call the class of languages that can be recognised by these automata *ω-regular languages*. The *parity index* of L is the minimal number k such that L can be recognised by a deterministic parity automaton using k output colours (which coincides with the minimal number of colours used by a Muller automaton recognising L [10, Proposition 6.14]).

Typeness. Let $\mathcal{A}_1 = (Q, q_{\mathsf{init}}, \Sigma, \Delta, \Gamma_1, \mathsf{col}_1, W_1)$ be a deterministic automaton, and let $\mathcal{C}$ be a class of languages (potentially containing languages over different alphabets). We say that $\mathcal{A}_1$ can be *relabelled* with a $\mathcal{C}$-acceptance condition, or that $\mathcal{A}$ is $\mathcal{C}$*-type*, if there is $W_2 \subseteq \Gamma_2^\omega$, $W_2 \in \mathcal{C}$, and a colouring function $\mathsf{col}_2 \colon \Delta \to \Gamma_2$ such that $\mathcal{A}_1$ is equivalent to $\mathcal{A}_2 = (Q, q_{\mathsf{init}}, \Sigma, \Delta, \Gamma_2, \mathsf{col}_2, W_2)$. In this case, we say that (col_1, W_1) and (col_2, W_2) are *equivalent acceptance conditions over* $\mathcal{A}$.

Given a Muller automaton $\mathcal{A}$, we use the expression *deciding the typeness* of $\mathcal{A}$ for the problem of answering if $\mathcal{A}$ is Rabin type, Streett type, parity type, or none of those.[1]

▶ Remark 3. In this work, we only consider typeness for deterministic automata for simplicity. For non-deterministic models, typeness admits two non-equivalent definitions [26]: (1) the acceptance status of each individual infinite path coincide for both acceptance conditions, or (2) both automata recognise the same language.

▶ **Example 4.** The automaton $\mathcal{A}_2$ from Figure 1 is Rabin type, as we have labelled it with a Rabin acceptance condition that is equivalent over $\mathcal{A}$ to the Muller condition given by $\mathcal{F}$ (in this case, both conditions use the same set of colours $\Gamma = \{\alpha, \beta, \gamma\}$). However, we note that $\mathsf{Rabin}_\Gamma(\mathcal{R}) \neq \mathsf{Muller}_\Gamma(\mathcal{F})$, as $\gamma^\omega \in \mathsf{Rabin}_\Gamma(\mathcal{R})$, while $\gamma^\omega \notin \mathsf{Muller}_\Gamma(\mathcal{F})$. This is possible, as no infinite path in $\mathcal{A}_2$ is labelled by a word that differentiates both languages (such as γ^ω).

2.2 The Zielonka tree and the Zielonka DAG

We represent *trees* and *directed acyclic graphs* (DAGs) as pairs $T = (N, \preceq)$ with N a non-empty finite set of nodes and $\preceq$ the *ancestor relation* ($n \preceq n'$ meaning that n is above n'). An *A-labelled tree* (resp. *A-labelled DAG*) is a tree (resp. DAG) together with a labelling function $\nu \colon N \to A$. We write $|T|$ to denote the number of nodes of a tree T.

▶ **Definition 5** ([42]). *Let $\mathcal{F} \subseteq 2^\Gamma_+$ be a family of non-empty subsets of a finite set Γ. The* Zielonka tree *for $\mathcal{F}$ (over Γ),*[2] *denoted $\mathcal{Z}_\mathcal{F} = (N, \preceq, \nu : N \to 2^\Gamma_+)$ is a 2^Γ_+-labelled tree with nodes partitioned into* round nodes *and* square nodes*, $N = N_{\bigcirc} \sqcup N_{\square}$, such that:*

- *The root is labelled Γ.*
- *If a node is labelled $X \subseteq \Gamma$, with $X \in \mathcal{F}$, then it is a round node, and it has a child for each maximal non-empty subset $Y \subseteq X$ such that $Y \notin \mathcal{F}$, which is labelled Y.*
- *If a node is labelled $X \subseteq \Gamma$, with $X \notin \mathcal{F}$, then it is a square node, and it has a child for each maximal non-empty subset $Y \subseteq X$ such that $Y \in \mathcal{F}$, which is labelled Y.*

▶ **Example 6.** Let $\mathcal{F} = \{\{\gamma, \alpha\}, \{\gamma, \beta\}, \{\beta\}\}$ be the Muller condition of the automaton from Example 2, over the alphabet $\{\alpha, \beta, \gamma\}$. In Figure 2 we show the Zielonka tree of $\mathcal{F}$.

The *Zielonka DAG* of a family $\mathcal{F} \subseteq 2^\Gamma_+$ is the labelled directed acyclic graph $\mathcal{Z}\text{-}\mathsf{DAG}_\mathcal{F}$ obtained by merging the nodes of $\mathcal{Z}_\mathcal{F}$ with a common label. It inherits the partition into round and square nodes, with children of round nodes being square and vice-versa.

▶ **Lemma 7** (Implied by [21, Lemma 1]). *Given a 2^Γ_+-labelled tree T with a partition into round and square nodes, we can decide in polynomial time whether there is a family $\mathcal{F} \subseteq 2^\Gamma_+$ such that $T = \mathcal{Z}_\mathcal{F}$. In the affirmative case, this family is uniquely determined.*

[1] We could consider further classes of acceptance conditions such as Büchi, coBüchi, generalised Büchi, weak, etc... We refer to [10, Appendix A] for more details on these acceptance types.

[2] The definition of $\mathcal{Z}_\mathcal{F}$, as well as most subsequent definitions, do not only depend on $\mathcal{F}$ but also on the alphabet Γ. Although this dependence is important, we do not explicitly include it in the notations in order to lighten them, as most of the times the alphabet will be clear from the context.

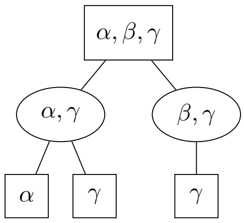


Figure 2 Zielonka tree $\mathcal{Z}_{\mathcal{F}}$ for $\mathcal{F} = \{\{\gamma, \alpha\}, \{\gamma, \beta\}, \{\beta\}\}$.

Representation of acceptance conditions. There is a wide variety of ways to represent a Muller language, including as its Zielonka tree, its Zielonka DAG, or *colour-explicitly*, that is, as a list of the subsets appearing in $\mathcal{F}$. In Figure 3 we summarise the relationship between these representations. We highlight that the Zielonka DAG can be built in polynomial time from both the Zielonka tree and from a colour-explicit representation of a Muller condition [23, Theorem 3.17]. The exponential-size separation between the Zielonka tree and colour-explicit representations, as well as explicit examples showing the gap between Zielonka trees and DAGs are original contributions.

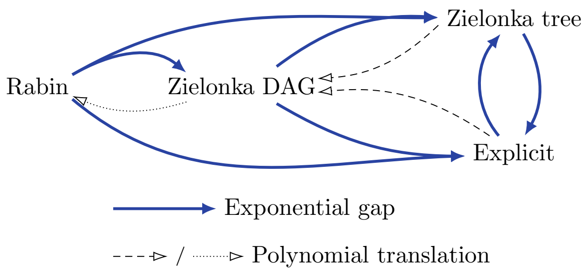


Figure 3 Comparison between the different representations of Muller conditions. A blue bold arrow from X to Y means that converting an X-representation into the form Y requires exponential time. A dashed arrow from X to Y means that a conversion can be computed in polynomial time. The dotted arrow indicates that the polynomial translation can only be applied on a fragment of X, as it is more expressive than Y.

2.3 The Alternating Cycle Decomposition

We now present the Alternating Cycle Decomposition and its DAG-version, following [10].

Let $\mathcal{A}$ be an automaton with Q and Δ as set of states and transitions, respectively. A *cycle* of $\mathcal{A}$ is a subset $\ell \subseteq \Delta$ such that there is a (not necessarily simple) path with the same starting and ending state such that the set of edges it visits is ℓ. The set of cycles of an automaton $\mathcal{A}$ is written $\mathit{Cycles}(\mathcal{A})$. We will consider the set of cycles ordered by inclusion. If we see $\mathcal{A}$ as a graph, its cycles are the strongly connected subgraphs of that graph, and the maximal cycles are its strongly connected components (SCCs). Let $\mathcal{A}$ be a Muller automaton with acceptance condition $(\mathsf{col}, \mathsf{Muller}_\Gamma(\mathcal{F}))$. Given a cycle $\ell \in \mathit{Cycles}(\mathcal{A})$, we say that ℓ is *accepting* (resp. *rejecting*) if $\mathsf{col}(\ell) \in \mathcal{F}$ (resp. $\mathsf{col}(\ell) \notin \mathcal{F}$).

▶ **Definition 8.** *Let $\ell_0 \in \mathit{Cycles}(\mathcal{A})$ be a cycle. We define the* tree of alternating subcycles *of ℓ_0, denoted $\mathsf{AltTree}(\ell_0)$, as a $\mathit{Cycles}(\mathcal{A})$-labelled tree with nodes partitioned into* round nodes *and* square nodes*, $N = N_{\bigcirc} \sqcup N_{\square}$, such that:*

- *The root is labelled* ℓ_0.
- *If a node is labelled* $\ell \in \mathcal{Cycles}(\mathcal{A})$*, and* ℓ *is an accepting cycle* ($\mathsf{col}(\ell) \in \mathcal{F}$)*, then it is a round node, and its children are labelled exactly with the maximal subcycles* $\ell' \subseteq \ell$ *such that* ℓ' *is rejecting* ($\mathsf{col}(\ell') \notin \mathcal{F}$).
- *If a node is labelled* $\ell \in \mathcal{Cycles}(\mathcal{A})$*, and* ℓ *is a rejecting cycle* ($\mathsf{col}(\ell) \notin \mathcal{F}$)*, then it is a square node, and its children are labelled exactly with the maximal subcycles* $\ell' \subseteq \ell$ *such that* ℓ' *is accepting* ($\mathsf{col}(\ell') \in \mathcal{F}$).

▶ **Definition 9** (Alternating cycle decomposition). *Let* $\mathcal{A}$ *be a Muller automaton, and let* $\ell_1, \ell_2, \dots, \ell_k$ *be an enumeration of its maximal cycles. We define the* alternating cycle decomposition *of* $\mathcal{A}$ *to be the forest* $\mathcal{ACD}(\mathcal{A}) = \{\mathsf{AltTree}(\ell_1), \dots, \mathsf{AltTree}(\ell_k)\}$.

▶ Remark 10. The Zielonka tree can be seen as the special case of the alternating cycle decomposition for an automata with a single state.

As mentioned in the introduction, the ACD was introduced in order to build small parity automata from Muller ones: given the ACD of a Muller automaton $\mathcal{A}$, we can build in polynomial time an equivalent parity automaton $\mathcal{P}_{\mathcal{A}}^{\mathsf{ACD}}$ (called the *ACD-parity transform* of $\mathcal{A}$) of minimal size amongst all automata obtained from $\mathcal{A}$ by "duplication of states". See [10, Section 5.2] for a formal statement and further results.

▶ **Example 11.** Figure 4 contains the alternating cycle decomposition of the automata $\mathcal{A}_1$ and $\mathcal{A}_2$ from Figure 1. We represent their transitions as pairs $(q, a) \in Q \times \Sigma$. Since both automata are strongly connected, each ACD consists in a single tree, whose root is the whole set of transitions.

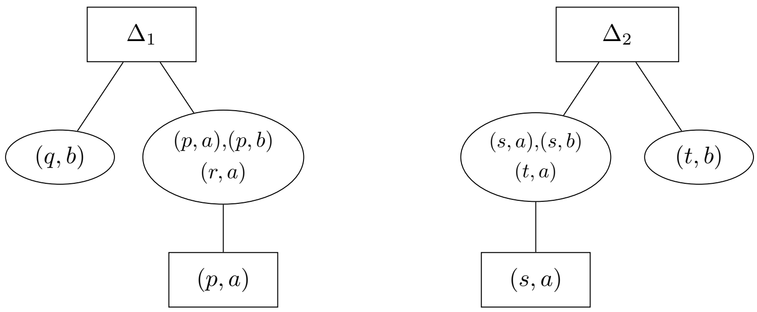


Figure 4 Alternating cycle decomposition of $\mathcal{A}_1$ (on the left) and $\mathcal{A}_2$ (on the right), from Figure 1.

The *ACD-DAG* of a Muller automaton $\mathcal{A}$, written $\mathcal{ACD}$-$\mathsf{DAG}(\mathcal{A})$, is the family of labelled DAGs obtained by merging nodes with the same label in the trees of $\mathcal{ACD}(\mathcal{A})$. It is useful for deciding the typeness and the parity index of $\mathcal{L}(\mathcal{A})$, as stated next.

▶ **Proposition 12** ([10, Section 6.1]). *Given a deterministic Muller automaton* $\mathcal{A}$ *and its ACD-DAG, we can decide the typeness of* $\mathcal{A}$ *and compute the parity index of* $\mathcal{L}(\mathcal{A})$ *in polynomial time.*

3 Computation of the Alternating Cycle Decomposition

We present in this section our central contribution: a polynomial-time algorithm to compute the alternating cycle decomposition of a Muller automaton with the acceptance condition given by a Zielonka tree (Theorem 13). This shows that the computation of the ACD is not harder than that of the Zielonka tree, (partially) explaining the strikingly performing experimental results from [12]. We also show that if the acceptance condition is represented as a Zielonka DAG, we can compute $\mathcal{ACD}$-DAG($\mathcal{A}$) in polynomial time (Theorem 15), from which we can derive decidability in polynomial time of typeness of Muller automata (Corollary 16).

3.1 Statements of the results

▶ **Theorem 13** (Computation of the ACD). *Given a Muller automaton $\mathcal{A}$ with acceptance condition represented by a Zielonka tree $\mathcal{Z}_\mathcal{F}$,*[3] *we can compute $\mathcal{ACD}(\mathcal{A})$ in polynomial time in $|\mathcal{A}| + |\mathcal{Z}_\mathcal{F}|$.*

As stated in the previous section, given the ACD of a Muller automaton $\mathcal{A}$, we can transform $\mathcal{A}$ in polynomial time into its ACD-parity-transform: a parity automaton equivalent to $\mathcal{A}$ that is minimal amongst parity automata obtained as a transformation of $\mathcal{A}$. The previous theorem implies that this can be done even if only the Zielonka tree of the acceptance condition of $\mathcal{A}$ is given as input, together with the automaton structure.[4]

▶ **Corollary 14.** *We can compute the ACD-parity-transform of a Muller automaton in polynomial time, if its acceptance condition is given by a Zielonka tree.*

▶ **Theorem 15** (Computation of the ACD-DAG). *Given a Muller automaton $\mathcal{A}$ with acceptance condition represented by a Zielonka DAG $\mathcal{Z}$-DAG$_\mathcal{F}$ (resp. colour-explicitly), we can compute $\mathcal{ACD}$-DAG($\mathcal{A}$) in polynomial time in $|\mathcal{A}| + |\mathcal{Z}\text{-DAG}_\mathcal{F}|$ (resp. $|\mathcal{A}| + |\mathcal{F}|$).*

Combining Theorem 15 with Propositions 12, we directly obtain that we can decide typeness of Muller automata and the parity index of their languages in polynomial time.

▶ **Corollary 16** (Polynomial-time decidability of typeness and parity index). *Given a deterministic Muller automaton $\mathcal{A}$ with its acceptance condition represented colour-explicitly, as a Zielonka tree, or as a Zielonka DAG, we can decide the typeness of $\mathcal{A}$, and determine the parity index of $\mathcal{L}(\mathcal{A})$, in polynomial time.*

The decidability of the parity index in polynomial time had already been obtained by Wilke and Yoo [41]. This result contrasts with the fact that deciding the parity index of a language represented by a deterministic Rabin or Streett automaton is NP-complete [25].

3.2 Main algorithm and complexity

Description of the algorithm. We describe an algorithm computing $\mathcal{ACD}$-DAG($\mathcal{A}$) from a Muller automaton $\mathcal{A}$. To obtain $\mathcal{ACD}(\mathcal{A})$, it suffices then to unfold this DAG. This algorithm builds the ACD-DAG in a top-down fashion: first, it computes the strongly connected components of $\mathcal{A}$ and initialises the root of each of the DAGs in $\mathcal{ACD}$-DAG($\mathcal{A}$). Then, it iteratively computes the children of the already found nodes as follows: Given a node n labelled ℓ (assume that ℓ is an accepting cycle), the algorithm goes through all square nodes m in the Zielonka DAG and for each of them computes the maximal sub-cycles of ℓ whose set of colours is included in the label of m, but not in those of any of its children. The algorithm then selects maximal cycles among those found, add them to $\mathcal{ACD}$-DAG($\mathcal{A}$) (if they do not already appear there) and sets them as children of n.

[3] Lemma 7 lets us check in polynomial time if a tree indeed is the Zielonka tree of a Muller condition.
[4] Also, we note that given $\mathcal{A}$ and its ACD, it is immediate to compute a Zielonka tree over the set of colours $\Gamma = \Delta$ defining an equivalent acceptance condition over $\mathcal{A}$.

Complexity analysis. We explain now how we obtain a polynomial upper bound on the complexity of the algorithms presented in the previous paragraph.

We first remark that we need to make at $|\mathcal{ACD}\text{-}\mathsf{DAG}(\mathcal{A})|$ computations of the children of a node, as each node of the ACD-DAG is considered at most once by the algorithm. Therefore, to obtain Theorem 15 (computation of the ACD-DAG) we need to show that:

1. We can compute the children of a node in polynomial time in $|Q| + |\mathcal{Z}\text{-}\mathsf{DAG}_{\mathcal{F}}|$, and
2. $|\mathcal{ACD}\text{-}\mathsf{DAG}(\mathcal{A})|$ is polynomial in $|Q| + |\mathcal{Z}\text{-}\mathsf{DAG}_{\mathcal{F}}|$.

To establish Theorem 13 (computation of the ACD), we remark that we can compute $\mathcal{ACD}(\mathcal{A})$ from $\mathcal{A}$ and $\mathcal{Z}_{\mathcal{F}}$ by simply folding $\mathcal{Z}_{\mathcal{F}}$ to obtain $\mathcal{Z}\text{-}\mathsf{DAG}_{\mathcal{F}}$, apply Theorem 15 to get $\mathcal{ACD}\text{-}\mathsf{DAG}(\mathcal{A})$, and then unfold the latter to obtain $\mathcal{ACD}(\mathcal{A})$. The first two steps require a time polynomial in $|\mathcal{Z}\text{-}\mathsf{DAG}_{\mathcal{F}}|+|Q| \leq |\mathcal{Z}_{\mathcal{F}}|+|Q|$, while the third step takes a time polynomial in $|\mathcal{ACD}(\mathcal{A})|$. Thus, to obtain the theorem, it suffices to establish

3. $|\mathcal{ACD}(\mathcal{A})|$ is polynomial in $|Q| + |\mathcal{Z}_{\mathcal{F}}|$.

The most technical part lies in the proofs of items 2 and 3, stated below.

▶ **Proposition 17.** *Let $\mathcal{A}$ be a Muller automaton and $\mathcal{F}$ the family defining its acceptance condition. Then,*

a) $|\mathcal{ACD}(\mathcal{A})| \leq |Q| \cdot |\mathcal{Z}_{\mathcal{F}}|$.

b) $|\mathcal{ACD}\text{-}\mathsf{DAG}(\mathcal{A})| \leq |Q| \cdot |\mathcal{Z}\text{-}\mathsf{DAG}_{\mathcal{F}}|$.

We describe now the main ideas of the proof of Proposition 17. We use the notion of *local subtree at a state* of the ACD. If q is a state of $\mathcal{A}$ appearing in the SCC ℓ_i, we define the *local subtree at* q, noted $\mathcal{T}_q$, as the subtree of $\mathsf{AltTree}(\ell_i)$ containing the nodes $N_q = \{n \in \mathsf{AltTree}(\ell_i) \mid q \text{ is a state in the label of } n\}$. We define analogously the *local subDAG of $\mathcal{ACD}$-$\mathsf{DAG}(\mathcal{A})$ at q*, noted $\mathcal{D}_q$.

We remark that $|\mathcal{ACD}(\mathcal{A})| \leq \sum_{q\in Q} |\mathcal{T}_q|$ (resp. $|\mathcal{ACD}\text{-}\mathsf{DAG}(\mathcal{A})| \leq \sum_{q\in Q} |\mathcal{D}_q|$), as each node of the ACD appears in some local subtree. Therefore, it suffices to bound the size of the local subtrees (resp. local subDAGs) to obtain a polynomial bound on the size of $\mathcal{ACD}(\mathcal{A})$ (resp. $\mathcal{ACD}\text{-}\mathsf{DAG}(\mathcal{A})$) to deduce Proposition 17. Quite surprisingly, the arguments to bound these objects are slightly different in each case.

▶ **Lemma 18.** *For every state q, the tree $\mathcal{T}_q$ has size at most $|\mathcal{Z}_{\mathcal{F}}|$.*

Proof sketch. We define in a top-down fashion an injective function $f\colon \mathcal{T}_q \to \mathcal{Z}_{\mathcal{F}}$. For the base case, we send the root of $\mathcal{T}_q$ to the root of $\mathcal{Z}_{\mathcal{F}}$. Let n be a node in $\mathcal{T}_q$ such that $f(n)$ has been defined, and let $n_1, \dots, n_k$ be its children. The key technical result is to show that there are k descendants of $f(n)$, containing the sets of labels of $n_1, \dots, n_k$, respectively, that are incomparable for the ancestor relation. Then, the subtrees rooted at these nodes are pairwise disjoint, which allows to define $f(n_i)$ for all i and carry out the induction. ◀

We conclude that the size of $\mathcal{ACD}(\mathcal{A})$ is polynomial in $|Q| + |\mathcal{Z}_{\mathcal{F}}|$, deriving the first item of Proposition 17:

$$|\mathcal{ACD}(\mathcal{A})| \leq \sum_{q\in Q} |\mathcal{T}_q| \leq |Q| \cdot |\mathcal{Z}_{\mathcal{F}}|.$$

▶ **Lemma 19.** *For every state q, the DAG $\mathcal{D}_q$ has size at most $|\mathcal{Z}\text{-}\mathsf{DAG}_{\mathcal{F}}|$.*

Proof sketch. As before, we define an injective function $f\colon \mathcal{D}_q \to \mathcal{Z}\text{-}\mathsf{DAG}_{\mathcal{F}}$. However, now we cannot use the fact that the subDAGs rooted at k incomparable elements are disjoint.

To circumvent this difficulty, for each node n in $\mathcal{D}_q$ different from the root, we fix an arbitrary immediate ancestor of n, noted $\mathsf{pred}(n)$ (that is, n is a child of $\mathsf{pred}(n)$). For a node n in $\mathcal{D}_q$, we let C_n be the set of colours appearing in the label of n. We define f recursively:

For the root n_0 of $\mathcal{D}_q$, we let $f(n_0)$ be a maximal (deepest) node in $\mathcal{Z}$-$\mathsf{DAG}_{\mathcal{F}}$ containing C_{n_0} in its label. For n a node such that we have define f for all its ancestors, we let $f(n)$ be a maximal node in the subDAG rooted at $f(\mathsf{pred}(n))$ containing C_n in its label (we note that $f(n)$ is a round node if and only if n is a round node). The most technical part of the proof is to show injectivity of the obtained function. ◀

4 Minimisation of colours and Rabin pairs

We consider the problem of minimising the representation of the acceptance condition of automata. That is, given a deterministic automaton $\mathcal{A}$ using a Muller (resp. Rabin) acceptance condition, what is the minimal number of colours (resp. Rabin pairs) needed to define an equivalent acceptance condition over $\mathcal{A}$?

We first study the minimisation of colours for Muller languages, without taking into account the structure of the automaton. We show that given the Zielonka DAG of the condition (resp. set of Rabin pairs), we can minimise its number of colours (resp. number of Rabin pairs) in polynomial time (Theorems 20 and 21). We provide an alternative point of view over the minimisation of Rabin pairs, using so-called generalised Horn formulas (see Remark 23). Then, we tackle the same question taking into account the structure of the automaton. Surprisingly, we show that in this case both problems are NP-complete, even if the ACD is given as input (Theorems 26 and 27).

4.1 Minimisation of the representation of Muller languages in PTIME and generalised Horn formulas

Minimisation of colours for Muller languages. We say that a Muller language $\mathsf{Muller}_\Sigma(\mathcal{F})$ is *k-colour type* if there is a set of k colours Γ, a family of sets $\mathcal{F}' \subseteq 2^\Gamma_+$ and a mapping $\phi\colon \Sigma \to \Gamma$ such that for all $S \in 2^\Sigma_+$, $S \in \mathcal{F} \iff \phi(S) \in \mathcal{F}'$.

Note that this is equivalent to asking if all automata using $\mathsf{Muller}_\Sigma(\mathcal{F})$ as acceptance condition can be relabelled with an equivalent Muller condition using at most k colours. (However, it is *not* the same as having a Muller automaton recognising $\mathsf{Muller}_\Sigma(\mathcal{F})$ using at most k colours.)

Colour-Minimisation-ML is the problem of deciding whether a given Muller language (represented by its Zielonka DAG) is k-colour type. We chose to specify the input as a Zielonka DAG, as it is more succinct than the other representations we consider (c.f. Figure 3). We now prove that this problem can be solved in polynomial time, which implies that it can be equally solved in polynomial time if the Muller language is represented colour-explicitly, or as a Zielonka tree.

▶ **Theorem 20** (Tractability of minimisation of colours for Muller languages). *The problem* Colour-Minimisation-ML *can be solved in polynomial time.*

Proof sketch. We define two colours $a, b \in \Sigma$ as equivalent if for every node n of $\mathcal{Z}$-$\mathsf{DAG}_{\mathcal{F}}$, $a \in \nu(n) \iff b \in \nu(n)$. It is not difficult to see that we can merge equivalent colours, that is, we can define $\mathsf{Muller}(\mathcal{F})$ using as many colours as the number of classes for this equivalence relation. We prove that this is optimal: If $\mathsf{Muller}(\mathcal{F})$ can be defined using a mapping $\phi\colon \Sigma \to \Gamma$, then, for all $\alpha \in \Gamma$, the colours in $\phi^{-1}(\alpha)$ are equivalent. Therefore, it suffices to inspect $\mathcal{Z}$-$\mathsf{DAG}_{\mathcal{F}}$ to determine the number of equivalence classes. ◀

Minimisation of Rabin pairs for Rabin languages. In this section we tackle the minimisation of the number of Rabin pairs to represent Rabin languages. We provide a polynomial-time algorithm which turns a family of Rabin pairs into an equivalent one with a minimal number of pairs. The algorithm comes down to partially computing the Zielonka tree of the input Rabin language from top to bottom, and stopping when we can infer from the partial view of the tree a set of Rabin pairs equivalent to the input. We present the algorithm differently to clarify the proofs, in particular the proof that the resulting number of pairs is minimal.

We say that a Rabin language $L \subseteq \Sigma^\omega$ is *k-Rabin-pair type* if there is a family of k Rabin pairs $\mathcal{R}$ over some set of colours Γ and a mapping $\phi\colon \Sigma \to \Gamma$ such that for all $w \in \Sigma^\omega$, $w \in L \iff \phi(w) \in \mathsf{Rabin}_\Gamma(\mathcal{R})$.

Rabin-Pair-Minimisation-ML is the problem of deciding whether a language $\mathsf{Rabin}_\Sigma(\mathcal{R})$ (represented by the Rabin pairs R) is k-Rabin-pair type.

▶ **Theorem 21** (Tractability of minimisation of Rabin pairs for Rabin languages). *The problem* Rabin-Pair-Minimisation-ML *can be solved in polynomial time.*

We obtain the minimal set of Rabin pairs iteratively. We start with an empty set of pairs. While our set of pairs is not equivalent to the input one, we compute a maximal set of colours S accepted by the input set of pairs and not by our current set of pairs. We then compute the maximal subset T of S that is rejected by the input set of pairs. We infer from them a new Rabin pair, which accepts the sets of colours contained in S but not in T. We add this pair to our set of pairs.

We prove that the resulting set is optimal by showing that at all times, we can define an injective function from our current set of pairs to any set of pairs equivalent to the input.

The minimisation of pairs for Streett conditions in polynomial time follows by symmetry.

Generalised Horn formulas. We discuss an alternative point of view on the minimisation of Rabin pairs, via a generalisation of Horn formulas.

Horn formulas are a popular fragment of propositional logic, as they enjoy some convenient complexity properties. It is well-known that the satisfiability problem for those formulas can be solved in linear time [15].

We consider a succinct representation of Horn formulas, called generalised Horn formula. They allow one to merge several Horn clauses with the same premises, e.g. $(x_1 \wedge x_2 \implies y_1)$ and $(x_1 \wedge x_2 \implies y_2)$, into a single clause $(x_1 \wedge x_2 \implies y_1 \wedge y_2)$. We can apply the classical linear-time algorithm for satisfiability on this generalised form, however, note that it is not linear in the size of the generalised formula, but in the size of the implicit Horn formula represented.

▶ **Definition 22.** *A* generalised Horn clause *(or GH clause) is a Boolean formula of the form either $(x_1 \wedge \cdots \wedge x_n) \implies (y_1 \wedge \cdots \wedge y_m)$ or $(x_1 \wedge \cdots \wedge x_n) \implies \bot$ (in the latter case, the clause is called* negative*). A* generalised Horn formula *(or GH formula) is a conjunction of GH clauses. It is* simple *if none of its GH clauses are negative.*

▶ Remark 23 (Correspondence simple GH formulas $\leftrightarrow$ Streett conditions). We observe that there is a correspondence between simple GH formulas and Streett conditions. Define the function α that turns a GH clause $(x_1 \wedge \cdots \wedge x_n) \implies (y_1 \wedge \cdots \wedge y_m)$ into the Rabin pair $(\{y_1, \ldots, y_m\}, \{x_1, \ldots, x_n\})$. We extend it into a function turning simple GH formulas into families of Rabin pairs by defining $\alpha(\bigwedge_{i=1}^k \mathrm{GH}_i) = (\alpha(\mathrm{GH}_i))_{i=1}^k$. We can then observe that α is a bijection (we consider Boolean formulas up to commutation of the terms, for instance we consider that $\varphi \vee \psi$ and $\psi \vee \varphi$ are the same formula). We also note that the number of clauses of a simple GH formula is the number of pairs of its image by α.

Note that for all simple GH formula φ, the set of sets accepted by the Streett condition $\alpha(\varphi)$ is $\{\nu^{-1}(\bot) \mid \nu : \mathrm{Var} \to \{\top, \bot\}$ is a valuation satisfying $\varphi\}$. As a result, two simple GH formula are equivalent if and only if their images by α define the same Streett language.

As a consequence of this correspondence and Theorem 21, we obtain that we can minimise the number of clauses in a GH formula in polynomial-time. This result contrasts nicely with the NP-completeness of minimising the number of clauses in a Horn formula [4] (see also [14]). On the other hand, minimising the number of literals in a GH formula remains NP-complete, just like in the case of Horn formulas [20]. This can be showed by a slight adaptation of the reduction from [14] to GH formulas.

▶ **Proposition 24.** *There is a polynomial-time algorithm to minimise the number of clauses of a GH formula.*

Proof sketch. The polynomial-time minimisation of simple GH formulas follows from Theorem 21 and Remark 23. The extension to all Generalised Horn formulas is essentially a technicality, due to the fact that negative clauses cannot be directly translated into Rabin pairs as in the previously. We circumvent this problem by replacing them with some non-negative clauses and proving that minimising the initial formula comes down to minimising the resulting simple one. ◀

On the other hand, generalised Horn formulas are likely not a suitable representation for acceptance conditions on automata, as they yield an NP-complete emptiness problem (Proposition 25). This is an interesting example of a family of acceptance conditions whose satisfiability problem is in PTIME but which yields an NP-complete emptiness problem on automata.

▶ **Proposition 25.** *Checking emptiness of an automaton with an acceptance condition represented by a GH formula is* NP*-complete.*

Proof sketch. The NP upper bound follows from the one on Emerson-Lei conditions. For the hardness, we reduce from the Hamiltonian cycle problem. ◀

4.2 Minimisation of acceptance conditions on top of an automaton

We now consider the problem of minimising the number of colours or Rabin pairs used by a Muller or Rabin condition over a fixed automaton. We could expect that it is possible to generalise the previous polynomial time algorithms by using the ACD, instead of the Zielonka DAG. Quite surprisingly, we show that these problems become NP-complete when taking into account the structure of the automata.

Minimisation of colours on top of a Muller automaton. We say that a deterministic Muller automaton $\mathcal{A}$ is *k-colour type* if we can relabel it with a Muller condition using at most k output colours that is equivalent over $\mathcal{A}$ (and uses a single colour per edge). We also consider automata with multiple colours per edge (in this section, multiple labels may be relevant). We will nevertheless show that allowing them does not change the theoretical complexity of the problem. We say that $\mathcal{A}$ is *k-multi-colour type* if we can relabel it with an equivalent Muller condition using at most k colours, with possibly several colours per edge.

Colour-Minimisation-Aut (resp. Multi-Colour-Minimisation-Aut) is the problem of deciding whether a deterministic Muller automaton is k-colour type (resp. k-multi-colour type). These problems admit different variants according to the representation of the Muller condition. We will show that for the three representations we are concerned with (colour-explicit, Zielonka tree and Zielonka DAG), both problems are NP-complete. This implies that they are NP-hard even if the ACD is provided as input, by Theorem 13. Hugenroth showed[5] that, *for state-based automata*, the problem Colour-Minimisation-Aut is NP-hard when the acceptance condition of $\mathcal{A}$ is represented colour-explicitly or as a Zielonka tree [21]. However, it is not straightforward to generalise it to *transition-based automata*, since the classic translation between state-based and transition-based automata does not preserve minimality.

▶ **Theorem 26** (NP-completeness of minimisation of colours for Muller automata)**.** *The problems* Colour-Minimisation-Aut *and* Multi-Colour-Minimisation-Aut *are* NP*-complete, if the acceptance condition* $\mathsf{Muller}_\Gamma(\mathcal{F})$ *of* $\mathcal{A}$ *is represented colour-explicitly, as a Zielonka tree, Zielonka DAG or as the ACD of* $\mathcal{A}$*.*

To obtain the NP-hardness, we reduce from the chromatic number problem for graphs. We note that the fact that these problems lie in NP is not obvious: we could be tempted to guess an acceptance condition on the same automaton structure and check equivalence of the two automata. However, reducing the number of colours might blow up the size of the representation of the acceptance condition.

NP-upper bound: Proof sketch. We guess a colouring $\mathsf{col}' : \Delta \to [k]$ and check in polynomial time that there exists a family $\mathcal{F}'$ over $[k]$ defining an equivalent condition over $\mathcal{A}$. To do so, we remark that such $\mathcal{F}'$ exists if and only if there is no pair of words $w_+ \in \mathcal{L}(A)$ and $w_- \notin \mathcal{L}(\mathcal{A})$ such that the sets of colours produced infinitely often under col' by their runs are equal. The existence of such words reduces to emptiness of adequate Streett automata. ◀

Minimisation of Rabin pairs on top of a Rabin automaton. We say that a deterministic Muller automaton $\mathcal{A}$ is *k-Rabin-pair type* if we can relabel it with an equivalent Rabin condition using at most k Rabin pairs.

Rabin-Pair-Minimisation-Aut is the problem of deciding whether a given deterministic Rabin automaton is k-Rabin-pair type. As before, we can consider different representations of the acceptance condition of the automaton: using Rabin pairs, colour-explicitly, the Zielonka tree, the Zielonka DAG or by providing the ACD.

▶ **Theorem 27** (NP-completeness of minimisation of Rabin pairs for Rabin automata)**.** *The problem* Rabin-Pair-Minimisation-Aut *is* NP*-complete for all the previous representations of the acceptance condition.*

5 Conclusion

In this work we obtained several positive results concerning the complexity of simplifying the acceptance condition of an ω-automaton. Our first technical result is that the computation of the ACD (resp. ACD-DAG) of a Muller automaton is not harder than the computation of the Zielonka tree (resp. Zielonka DAG) of its acceptance condition (Theorems 13 and 15). This provides support for the assertion that the optimal transformation into parity automata based on the ACD is applicable in practical scenarios, backing the experimental evidence provided by the implementations of the ACD-transform [12].

Furthermore, this result has several implications for our simplification purpose: We can decide the typeness of Muller automata and compute the parity index of their languages in polynomial time (Corollary 16). In addition, we showed that we can minimise in polynomial

[5] As of today, the proof is not currently publicly available online, we got access to it by a personal communication. The theorem only expresses the NP-hardness for the colour-explicit representation, but a look into the reduction works unchanged if the condition is given as a Zielonka tree.

time the colours and Rabin pairs necessary to represent a Muller language. However, these problems become NP-hard when taking into account the structure of a particular automaton using this acceptance condition, even if the ACD of the automaton is provided as input. Nevertheless, we believe that the methods for the minimisation of colours in the case of Muller languages could be combined with the structure of the ACD to obtain heuristics reducing the number of colours used by Muller automata, which might lead to substantial (although not optimal) reductions.

In sum, our results clarify the potential of the ACD and complete our understanding of the complexity of simplifying the acceptance conditions of ω-automata.

References

1 Tomáš Babiak, František Blahoudek, Alexandre Duret-Lutz, Joachim Klein, Jan Křetínský, David Müller, David Parker, and Jan Strejček. The Hanoi omega-automata format. In *CAV*, pages 479–486, 2015. doi:10.1007/978-3-319-21690-4_31.

2 Christel Baier, František Blahoudek, Alexandre Duret-Lutz, Joachim Klein, David Müller, and Jan Strejček. Generic emptiness check for fun and profit. In *ATVA*, pages 445–461, 2019. doi:10.1007/978-3-030-31784-3_26.

3 Udi Boker. Why these automata types? In *LPAR*, volume 57 of *EPiC Series in Computing*, pages 143–163, 2018. doi:10.29007/c3bj.

4 Endre Boros and Ondřej Čepek. On the complexity of Horn minimization. *Rutgers University. Rutgers Center for Operations Research [RUTCOR]*, 1994.

5 J. Richard Büchi. On a decision method in restricted second order arithmetic. *Proc. Internat. Congr. on Logic, Methodology and Philosophy of Science*, pages 1–11, 1962.

6 Cristian S. Calude, Sanjay Jain, Bakhadyr Khoussainov, Wei Li, and Frank Stephan. Deciding parity games in quasi-polynomial time. *SIAM Journal on Computing*, 51(2):STOC17–152–STOC17–188, 2022. doi:10.1137/17M1145288.

7 Olivier Carton and Ramón Maceiras. Computing the Rabin index of a parity automaton. *RAIRO*, pages 495–506, 1999. doi:10.1051/ita:1999129.

8 Antonio Casares. *Structural properties of automata over infinite words and memory for games (Propriétés structurelles des automates sur les mots infinis et mémoire pour les jeux)*. Phd thesis, Université de Bordeaux, France, 2023. URL: https://theses.hal.science/tel-04314678.

9 Antonio Casares, Thomas Colcombet, and Nathanaël Fijalkow. Optimal transformations of games and automata using Muller conditions. In *ICALP*, volume 198, pages 123:1–123:14, 2021. doi:10.4230/LIPIcs.ICALP.2021.123.

10 Antonio Casares, Thomas Colcombet, Nathanaël Fijalkow, and Karoliina Lehtinen. From Muller to Parity and Rabin Automata: Optimal Transformations Preserving (History) Determinism. *TheoretiCS*, Volume 3, April 2024. doi:10.46298/theoretics.24.12.

11 Antonio Casares, Thomas Colcombet, and Karoliina Lehtinen. On the size of good-for-games Rabin automata and its link with the memory in Muller games. In *ICALP*, volume 229, pages 117:1–117:20, 2022. doi:10.4230/LIPIcs.ICALP.2022.117.

12 Antonio Casares, Alexandre Duret-Lutz, Klara J. Meyer, Florian Renkin, and Salomon Sickert. Practical applications of the Alternating Cycle Decomposition. In *TACAS*, volume 13244 of *Lecture Notes in Computer Science*, pages 99–117, 2022. doi:10.1007/978-3-030-99527-0_6.

13 Antonio Casares and Corto Mascle. The complexity of simplifying ω-automata through the alternating cycle decomposition. *CoRR*, abs/2401.03811, 2024. doi:10.48550/arXiv.2401.03811.

14 Tom Chang. Horn formula minimization. Master's thesis, Rochester Institute of Technology, 2006. Available at https://scholarworks.rit.edu/cgi/viewcontent.cgi?article=7895&context=theses.

15 William F Dowling and Jean H Gallier. Linear-time algorithms for testing the satisfiability of propositional Horn formulae. *The Journal of Logic Programming*, 1(3):267–284, 1984.

16 Alexandre Duret-Lutz, Etienne Renault, Maximilien Colange, Florian Renkin, Alexandre Gbaguidi Aisse, Philipp Schlehuber-Caissier, Thomas Medioni, Antoine Martin, Jérôme Dubois, Clément Gillard, and Henrich Lauko. From Spot 2.0 to Spot 2.10: What's new? In *CAV*, volume 13372 of *Lecture Notes in Computer Science*, pages 174–187, 2022. doi:10.1007/978-3-031-13188-2_9.

17 Stefan Dziembowski, Marcin Jurdziński, and Igor Walukiewicz. How much memory is needed to win infinite games? In *LICS*, pages 99–110, 1997. doi:10.1109/LICS.1997.614939.

18 E. Allen Emerson, Charanjit S. Jutla, and A. Prasad Sistla. On model-checking for fragments of μ-calculus. In *CAV*, volume 697 of *Lecture Notes in Computer Science*, pages 385–396, 1993. doi:10.1007/3-540-56922-7_32.

19 Javier Esparza, Jan Křetínský, Jean-François Raskin, and Salomon Sickert. From LTL and limit-deterministic Büchi automata to deterministic parity automata. In *TACAS*, pages 426–442, 2017. doi:10.1007/978-3-662-54577-5_25.

20 Peter L Hammer and Alexander Kogan. Optimal compression of propositional Horn knowledge bases: complexity and approximation. *Artificial Intelligence*, 64(1):131–145, 1993.

21 Christopher Hugenroth. Zielonka DAG acceptance and regular languages over infinite words. In *DLT*, volume 13911 of *Lecture Notes in Computer Science*, pages 143–155, 2023. doi:10.1007/978-3-031-33264-7_12.

22 Paul Hunter and Anuj Dawar. Complexity bounds for regular games. In *MFCS*, pages 495–506, 2005. doi:10.1007/11549345_43.

23 Paul Hunter and Anuj Dawar. Complexity bounds for Muller games. *Theoretical Computer Science (TCS)*, 2008.

24 Swen Jacobs, Guillermo A. Perez, Remco Abraham, Veronique Bruyere, Michael Cadilhac, Maximilien Colange, Charly Delfosse, Tom van Dijk, Alexandre Duret-Lutz, Peter Faymonville, Bernd Finkbeiner, Ayrat Khalimov, Felix Klein, Michael Luttenberger, Klara Meyer, Thibaud Michaud, Adrien Pommellet, Florian Renkin, Philipp Schlehuber-Caissier, Mouhammad Sakr, Salomon Sickert, Gaetan Staquet, Clement Tamines, Leander Tentrup, and Adam Walker. The reactive synthesis competition (SYNTCOMP): 2018-2021, 2022. arXiv:2206.00251.

25 Sriram C. Krishnan, Anuj Puri, and Robert K. Brayton. Structural complexity of omega-automata. In *STACS*, pages 143–156, 1995. doi:10.1007/3-540-59042-0_69.

26 Orna Kupferman, Gila Morgenstern, and Aniello Murano. Typeness for omega-regular automata. *Int. J. Found. Comput. Sci.*, 17(4):869–884, 2006. doi:10.1142/S0129054106004157.

27 Jan Křetínský, Tobias Meggendorfer, and Salomon Sickert. Owl: A library for ω-words, automata, and LTL. In *ATVA*, volume 11138 of *Lecture Notes in Computer Science*, pages 543–550, 2018. doi:10.1007/978-3-030-01090-4_34.

28 Christof Löding. Optimal bounds for transformations of ω-automata. In *FSTTCS*, pages 97–109, 1999. doi:10.1007/3-540-46691-6_8.

29 Michael Luttenberger, Philipp J. Meyer, and Salomon Sickert. Practical synthesis of reactive systems from LTL specifications via parity games. *Acta Informatica*, pages 3–36, 2020. doi:10.1007/s00236-019-00349-3.

30 Robert McNaughton. Testing and generating infinite sequences by a finite automaton. *Information and control*, 9(5):521–530, 1966. doi:10.1016/S0019-9958(66)80013-X.

31 Philipp Meyer and Salomon Sickert. On the optimal and practical conversion of Emerson-Lei automata into parity automata. Unpublished manuscript, obsoleted by the work of [9], 2021.

32 Philipp J. Meyer and Salomon Sickert. Modernising Strix. *SYNT Workshop*, 2021. URL: https://www7.in.tum.de/~sickert/publications/MeyerS21.pdf.

33 Thibaud Michaud and Maximilien Colange. Reactive synthesis from LTL specification with Spot. In *SYNT@CAV*, Electronic Proceedings in Theoretical Computer Science, 2018.

34 Andrzej W. Mostowski. Regular expressions for infinite trees and a standard form of automata. In *SCT*, pages 157–168, 1984. doi:10.1007/3-540-16066-3_15.

35 David Müller and Salomon Sickert. LTL to deterministic Emerson-Lei automata. In *GandALF*, pages 180–194, 2017. doi:10.4204/EPTCS.256.13.

36 Nir Piterman and Amir Pnueli. Faster solutions of Rabin and Streett games. In *LICS*,, pages 275–284. IEEE Computer Society, 2006. doi:10.1109/LICS.2006.23.

37 Amir Pnueli and Roni Rosner. On the synthesis of a reactive module. In *POPL*, pages 179–190, 1989. doi:10.1145/75277.75293.

38 Michael O. Rabin. Decidability of second-order theories and automata on infinite trees. *Transactions of the American Mathematical Society*, 141:1–35, 1969. URL: http://www.jstor.org/stable/1995086.

39 Tereza Schwarzová, Jan Strejček, and Juraj Major. Reducing acceptance marks in Emerson-Lei automata by QBF solving. In *SAT*, volume 271, pages 23:1–23:20, 2023. doi:10.4230/LIPIcs.SAT.2023.23.

40 Cong Tian and Zhenhua Duan. Büchi determinization made tighter. *CoRR*, abs/1404.1436, 2014. arXiv:1404.1436.

41 Thomas Wilke and Haiseung Yoo. Computing the Rabin index of a regular language of infinite words. *Inf. Comput.*, pages 61–70, 1996. doi:10.1006/inco.1996.0082.

42 Wiesław Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theoretical Computer Science*, 200(1-2):135–183, 1998. doi:10.1016/S0304-3975(98)00009-7.