arXiv:2401.03811v1 [cs.FL] 8 Jan 2024

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

Antonio Casares
LaBRI, Université de Bordeaux, France
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-hard 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 Automata minimisation, ω-regular languages, Alternating Cycle Decomposition

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.

Contents

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 [4, 41]. 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 [40] remains at the heart of the state-of-the-art LTL-synthesis tools [18, 31, 35, 37]. 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 [5] and parity games solvers are extremely performing in practice [25], while solving Rabin and Muller games is, respectively, NP-complete [17] and PSPACE-complete [23]. Moreover, many existing algorithms for solving these games are polynomial on the size of the game graph, and the exponential dependency is only on parameters coming from the acceptance condition: Muller games can be solved in time $\mathcal{O}(k^{5k}n^5)$ [5, 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 $\mathcal{O}(n^{r+3}rr!)$ [39, 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 $\mathcal{O}(2^k kn^2|\phi|)$ [1, 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 [45, 16]. Zielonka showed that we can use this structure to tell whether a Muller language can be expressed as a Rabin or a parity language [45, Section 5]. Moreover, it has been recently proved that the Zielonka tree provides minimal deterministic parity automata recognising a Muller condition [9, 33], 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 [24]. Recently, Hugenroth showed that many decision problems concerning Muller automata become tractable when using the Zielonka DAG to represent the acceptance condition [22].

The ACD: Theoretical applications. In 2021, Casares, Colcombet and Fijalkow [8] 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 that can be obtained by duplicating states of $\mathcal{A}$ [9, Theorem 5.32]. Moreover, the ACD (or its DAG-version) can be used to tell whether a Muller automaton can be relabelled with an acceptance condition of a simpler type [9, Section 6.1].

However, the works introducing the ACD [8, 9] are of theoretical nature, and no study of the computational 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 [15] and Owl 21.0 [27], and are used in the LTL-synthesis tools `ltlsynt` [35] and `STRIX` [31, 34] (top-ranked in the SYNTCOMP competitions [25]). In the tool paper [11], 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 [11, 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 manipulating automata. Therefore, given a Muller automaton $\mathcal{A}$ an important question is: Can we simplify the acceptance condition of $\mathcal{A}$ without adding further states? This question admits two slightly different interpretations:

Typeness problem. Can we relabel the acceptance condition of $\mathcal{A}$ 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 [9]. Also, it is a classical result that we can minimise in polynomial time the number of colours used by a parity automaton [6]. However, it was still unclear whether the ACD could be of any help for minimising 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 [42]. 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 [44]. Also, in their work about minimal history-deterministic Rabin automata, Casares, Colcombet and Lehtinen left open the question of the minimisation of Rabin pairs [10].

1.2 Contributions

We outline the main contributions of this work.

1. **Computation of the ACD and the ACD-DAG.** Our main contribution is to show that we can compute the ACD of a Muller automaton in polynomial time, provided that we are given the Zielonka tree of its acceptance condition as input (Theorem 3.1). 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 [11]. 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 3.3). The main technical challenge is to prove that the ACD (resp. ACD-DAG) has polynomial size in the size of the Zielonka tree (resp. Zielonka DAG).
2. **Deciding typeness and the parity index in polynomial time.** Combining the previous contributions with the results from [9], 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 3.4). We also obtain that we can decide the parity index of the language recognised by a Muller automaton in polynomial time (Corollary 3.5).
3. **Minimisation of colours and Rabin pairs of acceptance conditions.** Given a 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 4.2 and 4.5).
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-hard, even if the ACD is given as input (Theorems 4.13 and 4.15). This came as a surprise to us, as our first intuition was in fact that the ACD would allow to lift the previous polynomial-time minimisation results to the problem in which we take into account the structure of the automaton.
5. **Analysis on the size of different representations of Muller conditions.** We provide tight bounds on the size of the Zielonka tree in the worst case (Proposition 5.1). Combining them with [9, Theorem 4.13], we recover results from Löding [30] giving bounds on the size of deterministic parity automata, and extend them to history-deterministic automata. We moreover provide examples showing the exponential gap on the size of the different representations of Muller conditions (Section 5.2).

Furthermore, we include an appendix (Appendix A) in which we study a subclass of interest of boolean formulas, which we call generalised Horn formulas, and relate them to the problem of minimising the number of Rabin pairs of a Rabin language.

2 Preliminaries

Basic notations

For a set A we let $|A|$ denote its cardinality, 2^A its power set and $2^A_+ = 2^A \setminus \{\emptyset\}$. For a family of subsets $\mathcal{F} \subseteq 2^A$ and $A' \subseteq A$, we write $\mathcal{F}|_{A'} = \mathcal{F} \cap 2^{A'}$. For natural numbers $i \leq j$, $[i, j]$ stands for $\{i, i+1, \dots, j-1, j\}$.

For a set Σ, a word over Σ is a sequence of elements from Σ. The sets of finite and infinite words over Σ will be written Σ^* and Σ^ω, respectively, and we let $\Sigma^\infty = \Sigma^* \cup \Sigma^\omega$. Subsets of Σ^* and Σ^ω will be called languages. For a word $w \in \Sigma^\infty$ we write w_i to represent the i$^{\text{th}}$

letter of w. The concatenation of two words $u \in \Sigma^*$ and $v \in \Sigma^\infty$ is written $u \cdot v$, or simply uv. For a word $w \in \Sigma^\omega$, we let $\mathsf{Inf}(w) = \{a \in \Sigma \mid w_i = a$ for infinitely many $i \in \mathbb{N}\}$.

2.1 Automata over infinite words and their acceptance conditions

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 $u \in \Gamma^*$.

We say that $\mathcal{A}$ is *deterministic* (resp. *complete*) if for every $q \in Q$ and $a \in \Sigma$, there is at most (resp. at least) one transition of the form $q \xrightarrow{a} q'$.

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 $c_0c_1c_2 \cdots \in W$, 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

$$\mathcal{L}(\mathcal{A}) = \{w \in \Sigma^\omega \mid w \text{ is accepted by } \mathcal{A}\}.$$

We say that two automata over the same input alphabet are *equivalent* if they recognise the same language.

▶ Remark 2.1. Most results in this paper concern the set of accepting runs of automata, rather than their languages. For instance we will try to modify the acceptance condition while preserving the set of accepting runs. However in the case of deterministic complete automata, those two notions coincide: preserving the set of accepting runs is exactly the same as preserving the language. Hence all results pertaining to the languages recognised by automata appearing in this paper will concern deterministic automata.

▶ Remark 2.2 (Transition-based acceptance). We remark that the colours used to define the acceptance of runs appear *over transitions*, instead of over states. This makes an important difference for many decisions problems on automata over infinite words such as the ones considered in this paper. For a discussion on the differences between transition-based and state-based automata, and arguments on why the first should be preferred, we refer to [7, Chapter VI].

Some corollaries of our results will refer to history-deterministic automata, although this model will not play a central role in our work. An automaton $\mathcal{A}$ is *history-deterministic* if there is a function $\sigma\colon \Sigma^+ \to \Delta$ resolving its non-determinism in such a way that for every $w \in \mathcal{L}(\mathcal{A})$, the run built by this function is an accepting run.

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 $R = \{(\mathfrak{g}_1, \mathfrak{r}_1), \dots, (\mathfrak{g}_r, \mathfrak{r}_r)\}$ 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 $R = \{(\mathfrak{g}_1, \mathfrak{r}_1), \dots, (\mathfrak{g}_r, \mathfrak{r}_r)\}$ 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:

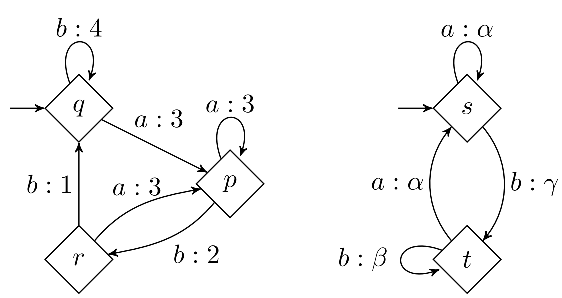
$$\mathsf{parity}_\Pi = \{w \in \Pi^\omega \mid \min \mathsf{Inf}(w) \text{ is even}\}.$$

A *Muller language* (resp. *Rabin/Streett language*) is a language that can be described as $\mathsf{Muller}_\Sigma(\mathcal{F})$ (resp. $\mathsf{Rabin}_\Gamma(\mathcal{R})/\mathsf{Streett}_\Gamma(\mathcal{R})$). 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 [2] for a more detailed account on different types of acceptance conditions.

▶ Remark 2.3. 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 2.4.** 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.



With a Muller condition
$\mathcal{F} = \{\{\gamma, \alpha\}, \{\beta, \gamma\}, \{\beta\}\}$.

With a Rabin condition
$\mathcal{R} = \{(\{\gamma\}, \{\beta\}), (\{\beta\}, \{\alpha\})\}$.

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 $L = \Sigma^* b^\omega + \Sigma^*(a^+ b)^\omega = \{w \in \Sigma^\omega \mid w \text{ has infinitely many } b\text{s and finitely many } abb \text{ factors}\}$.

The 8 classes of automata obtained by combining the 4 types of acceptance conditions above with deterministic and non-deterministic models are equally expressive [32, 36]. We call the class of languages that can be recognised by these automata *ω-regular languages*.

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* $\mathcal{A}$.

▶ Remark 2.5. In this work, we only consider typeness for deterministic automata. For non-deterministic models, typeness admits two non-equivalent definitions [29]: (1) the acceptance status of each individual infinite path coincide for both acceptance conditions, or (2) both automata recognise the same language.

▶ **Example 2.6.** 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.

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,
- $\mathcal{A}$ is Streett type, and
- $\mathcal{A}$ is parity type.

Formally, these are three different decision problems. We say that we can *decide the typeness of a class of Muller automata in polynomial time* if the three of them can be decided in polynomial time.[1]

Parity index

Let $L \subseteq \Sigma^\omega$ be an ω-regular language. 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.[2] Such number is well-defined, as any Muller automaton admits an equivalent deterministic parity automaton [36]. Moreover, it does not depend on the particular parity automaton used to recognise L:

▶ **Proposition 2.7** ([38]). *Let $\mathcal{A}$ be a deterministic parity automaton recognising a language $L \subseteq \Sigma^\omega$. If L has parity index k, then $\mathcal{A}$ admits an equivalent parity condition over it using only k output colours.*

As a matter of fact, the parity index of a language coincides with the minimal number of colours used by a Muller automaton recognising it [9, Proposition 6.14]. However, in contrast with the previous proposition, in order to reduce the number of colours of a Muller automaton we may need to modify its structure.

---

[1] Here, we could consider further classes of acceptance conditions such as Büchi, coBüchi, generalised Büchi, weak, etc... We refer to [9, Appendix A] for more details on these acceptance types. Our main result establishing decidability in polynomial time of typeness for Muller automata also holds for these acceptance conditions, as they are characterised by the ACD-DAG.

[2] This notion can be refined by taking into account whether the minimal colour needed is odd or even. We omit these details here for the sake of simplicit y of the presentation, and refer to [9, Definition 2.14] for formal definitions.

2.2 The Zielonka tree and the Zielonka DAG

We now introduce two closely-related ways of representing Muller conditions, the Zielonka tree and the Zielonka DAG, which are obtained by recursively listing the maximal accepting and rejecting subsets of colours of a family $\mathcal{F} \subseteq 2^{\Gamma}_{+}$.

Trees and DAGs

We represent a *tree* as a pair $T = (N, \preceq)$ with N a non-empty finite set of nodes and $\preceq$ the *ancestor relation* ($n \preceq n'$ meaning that n is above n'). We assume the reader to be familiar with the usual vocabulary associated with trees. The set of leaves of T is written $\mathsf{Leaves}(T)$. An *A-labelled tree* is a tree T together with a labelling function $\nu\colon N \to A$.

A *directed acyclic graph* (DAG) $(D, \preceq)$ is a non-empty finite set of nodes D equipped with an order relation $\preceq$ called the *ancestor relation* such that there is a minimal node for $\preceq$, called the *root*. We apply to DAGs similar vocabulary than for trees (children, leaves, depth, subDAG rooted at a node, ...). An *A-labelled DAG* is a DAG together with a labelling function $\nu\colon D \to A$.

The Zielonka tree

▶ **Definition 2.8** ([45]). *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),[3] 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.*

We write $|\mathcal{Z}_{\mathcal{F}}|$ to denote the number of nodes in $\mathcal{Z}_{\mathcal{F}}$.

▶ Remark 2.9. Let n be a node of $\mathcal{Z}_{\mathcal{F}}$ and let n_1 be a child of it. If $\nu(n_1) \subsetneq X \subseteq \nu(n)$, then $\nu(n_1) \in \mathcal{F} \iff X \notin \mathcal{F} \iff \nu(n) \notin \mathcal{F}$. In particular, if n_1, n_2 are two different children of n, then $\nu(n_1) \in \mathcal{F} \iff \nu(n_2) \in \mathcal{F} \iff \nu(n_1) \cup \nu(n_2) \notin \mathcal{F}$.

The next lemma provides a simple way to decide if a subset $C \subseteq \Gamma$ belongs to $\mathcal{F}$ given the Zielonka tree. It follows directly from the previous remark.

▶ **Lemma 2.10.** *Let $C \subseteq \Gamma$ and let n be a node of $\mathcal{Z}_{\mathcal{F}}$ such that $C \subseteq \nu(n)$ and that is maximal for $\preceq$ amongst nodes containing C in its label. Then, $C \in \mathcal{F}$ if and only if n is round.*

▶ **Example 2.11.** Let $\mathcal{F}$ be the Muller condition used by the automaton from Example 2.4: $\mathcal{F} = \{\{\gamma, \alpha\}, \{\gamma, \beta\}, \{\beta\}\}$, over the alphabet $\{\alpha, \beta, \gamma\}$. In Figure 2 we show the Zielonka tree of $\mathcal{F}$.

One important application of the Zielonka tree is that it provides minimal parity automata recognising Muller languages.

[3] 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.

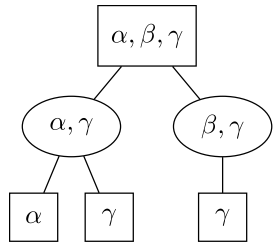


Figure 2 Zielonka tree $\mathcal{Z}_{\mathcal{F}}$ for $\mathcal{F} = \{\{\gamma, \alpha\}, \{\gamma, \beta\}, \{\beta\}\}$.

▶ **Proposition 2.12** ([9, Theorem 4.13]). *Let $L = \mathsf{Muller}_{\Sigma}(\mathcal{F})$ be a Muller language. There is a deterministic parity automaton of size $|\mathsf{Leaves}(\mathcal{Z}_{\mathcal{F}})|$ recognising L. Moreover, such an automaton is minimal both amongst deterministic and history-deterministic parity automata recognising L.*

The Zielonka tree can also be used to obtain minimal history-deterministic Rabin automata. The formal statement of this result can be found in [10, Proposition 11].

The Zielonka DAG

The *Zielonka DAG* of a family $\mathcal{F} \subseteq 2^{\Gamma}_{+}$ is the labelled directed acyclic graph obtained by merging the nodes of $\mathcal{Z}_{\mathcal{F}} = (N, \preceq, \nu)$ that share a common label. Formally, if is the labelled DAG $\mathcal{Z}\text{-}\mathsf{DAG}_{\mathcal{F}} = (N', \preceq', \nu')$ where $N' = \{C \subseteq \Sigma \mid \exists n$ node of the Zielonka tree such that $C = \nu(n)\}$, ν' is the identity and the relation $\preceq'$ is inherited from the ancestor relation of the tree: $C \preceq' D$ if there are n_C, n_D nodes of the Zielonka tree such that $\nu(n_C) = C$, $\nu(n_D) = D$ and $n_C \preceq n_D$. In particular, $C \preceq' D$ implies $D \subseteq C$ (but the converse does not hold in general).

We remark that $\mathcal{Z}\text{-}\mathsf{DAG}_{\mathcal{F}}$ inherits the partition of nodes into round and square ones. Moreover, children of a round node of the Zielonka DAG are square nodes and vice-versa. We also note that Remark 2.9 and Lemma 2.10 hold similarly replacing $\mathcal{Z}_{\mathcal{F}}$ by $\mathcal{Z}\text{-}\mathsf{DAG}_{\mathcal{F}}$ in their statement.

▶ **Example 2.13.** The Zielonka DAG of the condition $\mathcal{F} = \{\{\gamma, \alpha\}, \{\gamma, \beta\}, \{\beta\}\}$ is obtained by merging the nodes labelled $\{\gamma\}$ in the tree from Figure 2. Another example can be found in Figure 5 (page 30).

Representation of acceptance conditions

There is a wide variety of ways to *represent* a Muller language $W = \mathsf{Muller}_{\Gamma}(\mathcal{F})$, and the complexity and practicality of algorithms manipulating Muller automata may greatly differ depending on which of these representations is used [21, 23]. A *colour-explicit representation* is given simply as a list of the subsets appearing in $\mathcal{F} \subseteq 2^{\Gamma}_{+}$. In this section we have defined two further representations for Muller languages: the Zielonka tree and the Zielonka DAG. A thorough study of automata with acceptance condition given as a Zielonka DAG was conducted by Hugenroth [22]. Our results (of orthogonal nature) reinforce his thesis that the Zielonka DAG is a well-suited way of representing Muller acceptance conditions providing a good balance between succinctness and algorithmic properties.

In Figure 3 we provide a summary of the relationship between these three representations. These will be proved and studied in further detail in Section 5. 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 [24, Theorem 3.17], being the most succinct representation of the three.

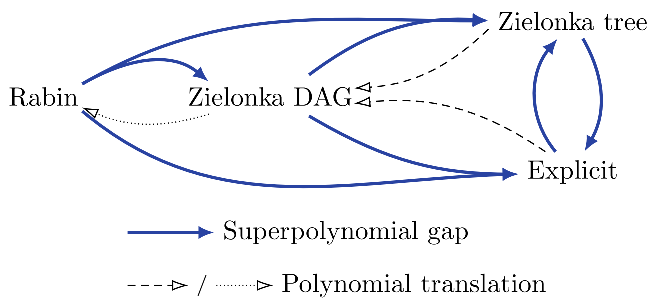


Figure 3 Comparison between the different representations of Muller conditions. A blue bold arrow from X to Y means that converting an X-representation into the form Y cannot be done in polynomial time. A dashed arrow from X to Y means the opposite. The dotted arrow indicates that the polynomial translation can only be applied on a fragment of X, as it is more expressive than Y.

In practical applications it is sometimes useful to have a more succinct representation, and a common choice are *Emerson-Lei conditions*, which describe a family $\mathcal{F}$ as a positive boolean formula over the primitives $\mathtt{Inf}(c)$ and $\mathtt{Fin}(c)$, for $c \in \Gamma$. We do not consider Emerson-Lei representations in this work, as they inherit the complexity analysis of boolean formulas; for instance, the problem of emptiness of Emerson-Lei automata (even with a single state) is essentially the SAT problem, and thus NP-complete.

2.3 The Alternating Cycle Decomposition

We now present the Alternating Cycle Decomposition and its DAG-version, following [9]. We also justify their interest by listing some key properties, mainly, optimal transformations of automata (Proposition 2.18) and characterisations of the typeness and the parity index of automata (Propositions 2.19 and 2.20). We start with some definitions about cycles of automata.

Cycles

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 finite path $q_0 \xrightarrow{a_0} q_1 \xrightarrow{a_1} q_2 \xrightarrow{a_2} \dots q_r \xrightarrow{a_r} q_0$ with $e_i = (q_i, a_i, q_{i+1}) \in \Delta$ and $\ell = \{e_0, e_1, \dots, e_r\}$. Note that we do not require this path to be simple, that is, edges and vertices may appear multiple times. The set of states of the cycle ℓ is $\mathsf{States}(\ell) = \{q_0, q_1, \dots q_r\}$. 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. For a state $q \in Q$, we note $\mathit{Cycles}_q(\mathcal{A})$ the subset of cycles of Q containing q. Note that $\mathit{Cycles}_q(\mathcal{A})$ is closed under union; moreover, the union of two cycles $\ell_1, \ell_2 \in \mathit{Cycles}(\mathcal{A})$ is again a cycle if and only if they have some state in common. A state is called *recurrent* if it belongs to some cycle and *transient* if it does not. 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. $\mathsf{col}(\ell) \notin \mathcal{F}$).

Tree of alternating subcycles and the Alternating Cycle Decomposition

▶ **Definition 2.14.** *Let $\ell_0 \in \mathcal{C}ycles(\mathcal{A})$ be a cycle. We define the tree of alternating subcycles of ℓ_0, denoted $\mathsf{AltTree}(\ell_0) = (N, \preceq, \nu\colon N \to \mathcal{C}ycles(\mathcal{A}))$ as a $\mathcal{C}ycles(\mathcal{A})$-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 $\ell \in \mathcal{C}ycles(\mathcal{A})$, 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 $\ell \in \mathcal{C}ycles(\mathcal{A})$, 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 2.15** (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 2.16. The Zielonka tree can be seen as the special case of the alternating cycle decomposition for automata with a single state. Indeed, a Muller language $\mathsf{Muller}_\Sigma(\mathcal{F})$ can be trivially recognised by a deterministic Muller automaton $\mathcal{A}$ with a single state q and self loops $q \xrightarrow{a:a} q$. The ACD of this automaton is exactly the Zielonka tree of $\mathcal{F}$.

▶ **Example 2.17.** We show the alternating cycle decomposition of the automata $\mathcal{A}_1$ and $\mathcal{A}_2$ from Figure 1 in Figure 4. As these automata are deterministic, we can represent their transitions as couples $(q, a) \in Q \times \Sigma$. Since both of them are strongly connected, each ACD consists in a single tree, whose root is the whole set of transitions. The bold coloured subtrees correspond to local subtrees at states p and t, respectively, as defined below.

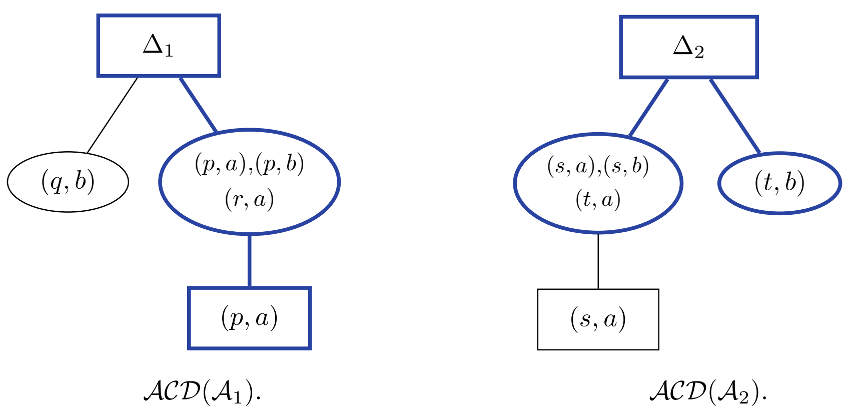


Figure 4 Alternating cycle decomposition of $\mathcal{A}_1$ and $\mathcal{A}_2$, from Figure 1. In bold blue, the local subtrees of $\mathcal{ACD}(\mathcal{A}_1)$ at p and of $\mathcal{ACD}(\mathcal{A}_2)$ at t.

Local subtrees

We remark that for a recurrent state q of $\mathcal{A}$, there is one and only one tree $\mathsf{AltTree}(\ell_i)$ in $\mathcal{ACD}(\mathcal{A})$ such that q appears in such a tree. On the other hand, transient vertices do not appear in the trees of $\mathcal{ACD}(\mathcal{A})$.

If q is a recurrent 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 $N_q = \{n \in \mathsf{AltTree}(\ell_i) \mid q \text{ is a state in } \nu(n)\}$. If q is a transient state, we define $\mathcal{T}_q$ to be a tree with a single node.

The ACD-parity-transform

As mentioned in the introduction, the ACD was introduced as a structure to build small parity automata from Muller ones. Casares, Colcombet and Fijalkow [8] defined the *ACD-parity-transform* $\mathcal{P}_{\mathcal{A}}^{\mathsf{ACD}}$ of a Muller automaton $\mathcal{A}$, which is an equivalent parity automaton with the property that it is minimal amongst parity automata that can be obtained from $\mathcal{A}$ by duplication of states. They formalise this minimality statement using morphisms of automata.[4]

Let $\mathcal{A}$ and $\mathcal{B}$ be two deterministic automata over the same input alphabet. A *morphism* $\varphi\colon \mathcal{A} \to \mathcal{B}$ is a function sending states of $\mathcal{A}$ to states of $\mathcal{B}$ such that:

- $\varphi(q_{\mathsf{init}}^{\mathcal{A}})$ is the initial state of $\mathcal{B}$,
- for all transition (q, a, q') in $\mathcal{A}$, $(\varphi(q), a, \varphi(q'))$ is a transition in $\mathcal{B}$, and
- a run ρ in $\mathcal{A}$ is accepting if and only if $\varphi(\rho)$ is accepting in $\mathcal{B}$.

▶ **Proposition 2.18** ([9, Section 5.2]). *Given a Muller automaton $\mathcal{A}$ and its ACD, we can build in polynomial time a parity automaton $\mathcal{P}_{\mathcal{A}}^{\mathsf{ACD}}$ equivalent to $\mathcal{A}$ such that no deterministic parity automaton admitting a morphism to $\mathcal{A}$ is smaller than $\mathcal{P}_{\mathcal{A}}^{\mathsf{ACD}}$.*

This optimality result has been generalised to history-deterministic parity automata, and optimal transformations towards history-deterministic Rabin automata based on the ACD have been also proposed [9].

The ACD-DAG

In the same way as we obtained the Zielonka DAG from the Zielonka tree, we define a DAG obtained by merging the nodes of the ACD sharing the same label.

Let $\mathcal{A}$ be a Muller automaton. The *DAG of alternating subcycles* of a cycle ℓ, denoted $\mathsf{AltDAG}(\ell)$ is the $\mathcal{C}ycles(\mathcal{A})$-labelled DAG obtained by merging the nodes of $\mathsf{AltTree}(\ell)$ with a same label. The *ACD-DAG* of a Muller automaton $\mathcal{A}$ is $\mathcal{ACD}\text{-}\mathsf{DAG}(\mathcal{A}) = \{\mathsf{AltDAG}(\ell_1), \dots, \mathsf{AltDAG}(\ell_k)\}$, where $\ell_1, \dots, \ell_k$ is an enumeration of the maximal cycles of $\mathcal{A}$ (that is, of its SCCs).

For q a recurrent state of $\mathcal{A}$, we define the *local subDAG at q*, noted $\mathcal{D}_q$, as the DAG obtained by merging the nodes of $\mathcal{T}_q$ with a same label. We note that if ℓ_i is the maximal cycle containing q, $\mathcal{D}_q$ coincides with the subDAG of $\mathsf{AltDAG}(\ell_i)$ consisting of the nodes labelled with cycles containing q.

The ACD-DAG of a deterministic Muller automaton $\mathcal{A}$ can be used to decide its typeness and the parity index of $\mathcal{L}(\mathcal{A})$.

▶ **Proposition 2.19** ([9, Section 6.1]). *Given a deterministic Muller automaton $\mathcal{A}$ and its ACD-DAG, we can decide the typeness of $\mathcal{A}$ in polynomial time. More precisely, $\mathcal{A}$ is:*

- *Rabin type if and if for all $q \in Q$ and round node $n \in \mathcal{D}_q$, n has at most one child in $\mathcal{D}_q$;*
- *Streett type if and if for all $q \in Q$ and square node $n \in \mathcal{D}_q$, n has at most one child in $\mathcal{D}_q$;*

[4] For simplicity, here we define only morphisms of deterministic automata. More general statements use the notions of locally bijective and history-deterministic morphisms. We refer to [9] for details.

- *parity type if and if for all $q \in Q$, $\mathcal{D}_q$ has a single branch.*

▶ **Proposition 2.20** ([9, Proposition 6.13]). *Let $\mathcal{A}$ be a deterministic Muller automaton. The parity index of $\mathcal{L}(\mathcal{A})$ coincides with the maximal height of a DAG from $\mathcal{ACD}$-DAG$(\mathcal{A})$ (which coincides with the maximal height of a tree from $\mathcal{ACD}(\mathcal{A})$).*

3 Computation of the Alternating Cycle Decomposition

We present in this section the main contribution of the paper: a polynomial-time algorithm to compute the alternating cycle decomposition of a Muller automaton, and its analysis. We prove that if the acceptance condition of the automaton is represented as a Zielonka tree, we can compute $\mathcal{ACD}(\mathcal{A})$ in polynomial time (Theorem 3.1). 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 [11]. 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 3.3), from which we can derive decidability in polynomial time of typeness of Muller automata (Corollary 3.4) and of the parity index of the languages they recognise (Corollary 3.5).

3.1 Statements of the results

We first state that results that will be obtained in this section. In all the section, given an automaton $\mathcal{A}$, Q will stand for its set of states.

▶ **Theorem 3.1** (Computation of the ACD). *Given a Muller automaton $\mathcal{A}$ with acceptance condition represented by a Zielonka tree $\mathcal{Z}_{\mathcal{F}}$, we can compute $\mathcal{ACD}(\mathcal{A})$ in polynomial time in $|Q| + |\mathcal{Z}_{\mathcal{F}}|$.*

As explained in Proposition 2.18, given the ACD of a Muller automaton $\mathcal{A}$, we can transform in polynomial time $\mathcal{A}$ 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.

▶ **Corollary 3.2.** *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 3.3** (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 $|Q| + |\mathcal{Z}\text{-DAG}_{\mathcal{F}}|$ (resp. $|Q| + |\mathcal{F}|$).*

Combining Theorem 3.3 with Propositions 2.19 and 2.20, we directly obtain that we can decide typeness of Muller automata and the parity index of their languages in polynomial time.

▶ **Corollary 3.4** (Polynomial-time decidability of typeness). *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 $\mathcal{A}$ in polynomial time.*

▶ **Corollary 3.5** (Polynomial-time decidability of 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 determine the parity index of $\mathcal{L}(\mathcal{A})$ in polynomial time.*

This last result contrasts with the fact that deciding the parity index of a language represented by a deterministic Rabin or Streett automaton is NP-complete [28, Theorem 28]. It was already well-known that the parity index was computable in polynomial time from a deterministic parity automata [38, 6].

3.2 Main algorithm

We present the pseudocode of an algorithm computing $\mathcal{ACD}\text{-DAG}(\mathcal{A})$ (Algorithm 1) from a Muller automaton $\mathcal{A}$. The full procedures requires a time polynomial in $|Q| + |\mathcal{Z}\text{-DAG}_{\mathcal{F}}| + |\mathcal{ACD}\text{-DAG}(\mathcal{A})|$; we will then obtain Theorem 3.3 by showing that $|\mathcal{ACD}\text{-DAG}(\mathcal{A})| \leq |Q| \cdot |\mathcal{Z}\text{-DAG}_{\mathcal{F}}|$ (if the acceptance condition is represented colour-explicitly, we can compute the Zielonka DAG from it in polynomial time [24]). If we want to compute the ACD of an automaton $\mathcal{A}$ with the acceptance condition given as a Zielonka tree, we can simply compute the Zielonka DAG from it, apply the previous procedure to get the ACD-DAG and then unfold the latter to obtain the ACD. As a result, we can compute the ACD in time polynomial in $|Q| + |\mathcal{Z}_{\mathcal{F}}| + |\mathcal{ACD}(\mathcal{A})|$; we will then obtain Theorem 3.1 by showing that $|\mathcal{ACD}(\mathcal{A})| \leq |Q| \cdot |\mathcal{Z}_{\mathcal{F}}|$. Quite surprisingly, the arguments we need to use to prove these upper bounds for $|\mathcal{ACD}\text{-DAG}(\mathcal{A})|$ and $|\mathcal{ACD}(\mathcal{A})|$ are quite different.

The algorithm we propose 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 $\mathcal{ACD}\text{-DAG}(\mathcal{A})$. Then, it iteratively computes the children of the already found nodes using the sub-procedure `ComputeChildren`, presented in Algorithm 2. Given a node n labelled with $\nu(n) = \ell$ (assume that ℓ is an accepting cycle), `ComputeChildren` goes through all round nodes in the Zielonka DAG and for each such node m computes the maximal sub-cycles of ℓ that whose set of colours is included in the one of m, but not in the one of any child of m. The algorithm then selects maximal cycles among all those, add them to $\mathcal{ACD}\text{-DAG}(\mathcal{A})$ (if they do not already appear in the DAG) and sets them as children of n.

We use the following notations:

- `SCC-Decomposition`$(\mathcal{S})$ outputs a list of the strongly connected components of $\mathcal{S}$.
- `pop`(stck) removes an element from the stack stck and returns it.
- `push`(stck, L) adds the elements of L to the stack stck.
- `MaxInclusion`(lst) returns the list of the maximal subsets in lst.

All the previous functions can be computed in polynomial time.

We provide in Algorithm 2 a procedure to compute the children of a node of the ACD-DAG. We show in Lemma 3.6 that this algorithm is correct and terminates in polynomial time in $|\mathcal{Z}\text{-DAG}_{\mathcal{F}}| + |Q|$. We say that two nodes have the *same shape* if they are both round or both square.

3.3 Complexity analysis

We now prove correctness and termination in polynomial time of the algorithms presented in the previous subsection, establishing Theorems 3.1 and 3.3.

We first remark that Algorithm 1 makes at most $|\mathcal{ACD}\text{-DAG}(\mathcal{A})|$ calls to the function `ComputeChildren`, as each node of the ACD-DAG is added at most once to nodesToTreat. Therefore, to obtain Theorem 3.3 (computation of the ACD-DAG) we need to show: (1) $|\mathcal{ACD}\text{-DAG}(\mathcal{A})|$ is polynomial in $|Q| + |\mathcal{Z}\text{-DAG}_{\mathcal{F}}|$, and (2) the function `ComputeChildren` takes polynomial time in this measure.

Algorithm 1 Computation of the ACD-DAG

Input: A Muller automaton $\mathcal{A}$
Output: $\mathcal{ACD}$-DAG($\mathcal{A}$)

$\langle \mathcal{S}_1, \dots, \mathcal{S}_r \rangle \leftarrow$ `SCC-Decomposition`($\mathcal{A}$)
Add $\mathcal{S}_1, \dots, \mathcal{S}_r$ as the root of r different DAGs of $\mathcal{ACD}$-DAG($\mathcal{A}$)
nodesToTreat $\leftarrow \langle \mathcal{S}_1, \dots, \mathcal{S}_r \rangle$ ▷ Initialise a stack
while nodesToTreat $\neq \emptyset$ **do**
    n $\leftarrow$ `pop`(nodesToTreat)
    children $\leftarrow$ `ComputeChildren`(n)
    newChildren $\leftarrow$ elements of children that do not appear in $\mathcal{ACD}$-DAG($\mathcal{A}$)
    Add the nodes of newChildren to $\mathcal{ACD}$-DAG($\mathcal{A}$)
    Add an edge from n to each element of children
    `push`(nodesToTreat, newChildren)
end while
return $\mathcal{ACD}$-DAG($\mathcal{A}$)

Algorithm 2 `ComputeChildren`(n): Computing the children of a node n of the ACD-DAG

Input: A node of $\mathcal{ACD}$-DAG($\mathcal{A}$) labelled by a cycle C
Output: Maximal subcycles $\ell_1, \dots, \ell_k$ of C such that $\mathsf{col}(\ell_i) \in \mathcal{F} \iff \mathsf{col}(C) \notin \mathcal{F}$.

children $\leftarrow \emptyset$
for $m \in \mathcal{Z}$-DAG$_{\mathcal{F}}$ a node of the same shape as n **do**
    $C_m \leftarrow$ restriction of C to transitions e such that $\mathsf{col}(e) \in C$
    $\langle C_{m,1}, \dots, C_{m,r} \rangle \leftarrow$ `SCC-Decomposition`(C_m)
    for $i = 1, \dots, r$ **do**
        if for all child p of m, $\mathsf{col}(C_{m,i}) \nsubseteq \mathsf{col}(\nu(p))$ **then**
            children $\leftarrow$ children $\cup \{C_{m,i}\}$
        end if
    end for
end for
children $\leftarrow$ `MaxInclusion`(children)
return children

We start by showing that we can compute the children of a node of $\mathcal{ACD}$-DAG($\mathcal{A}$) in polynomial time in $|Q| + |\mathcal{Z}\text{-DAG}_{\mathcal{F}}|$. The obtention of the upper bounds on $\mathcal{ACD}$-DAG($\mathcal{A}$) will be the subject of the next subsection.

▶ **Lemma 3.6.** *Algorithm 2 computes the list of children of a node of* $\mathcal{ACD}$-DAG($\mathcal{A}$) *in polynomial time in* $|\mathcal{Z}\text{-DAG}_{\mathcal{F}}| + |Q|$.

Proof. First let us argue that the returned list contains exactly the children of the input node in $\mathcal{ACD}$-DAG($\mathcal{A}$).

Let n be the input node, C its label, and let us assume that it is square, the other case is symmetric. Its children are the maximal cycles $\ell_1, \dots, \ell_k \in \mathit{Cycles}(C)$ such that $\mathsf{col}(\ell_i) \in \mathcal{F}$. By definition of $\mathcal{Z}\text{-DAG}_{\mathcal{F}}$, those are the maximal cycles such that there exists a round node m in $\mathcal{Z}\text{-DAG}_{\mathcal{F}}$ such that $\mathsf{col}(\ell_i) \subseteq C$ and $\mathsf{col}(\ell_i) \nsubseteq p$ for all children p of C. This is straightforwardly what Algorithm 2 computes, as the algorithm goes through all round nodes C, computes the maximal cycles whose set of colours are included in C but not in its children in $\mathcal{Z}\text{-DAG}_{\mathcal{F}}$ and adds them to children. It then outputs the maximal cycles in children.

For the complexity, note that we go through the for loop on line 2 at most $|\mathcal{Z}\text{-DAG}_{\mathcal{F}}|$ times, and through the for loop of line 5 at most $|Q|$ times at each iteration. Computing `SCC-Decomposition`($\mathcal{S}_C$) on line 4 requires time linear in $|Q|$ by Tarjan's algorithm [43]. As a result, the execution time of Algorithm 2 up to line 10 and the size of children after line 10 are both polynomial in $|Q| + |\mathcal{Z}\text{-DAG}_{\mathcal{A}}|$, hence the whole algorithm takes polynomial time in that measure. ◀

3.4 Upper bounds on the size of the ACD and the ACD-DAG

We now establish the desired upper bounds on the size of the ACD and the ACD-DAG. We start by proving that $|\mathcal{ACD}(\mathcal{A})| \leq |Q| \cdot |\mathcal{Z}_{\mathcal{F}}|$; the analysis of the size of $\mathcal{ACD}$-DAG($\mathcal{A}$) will be a refinement of this proof.

A polynomial upper bound on the size of $\mathcal{ACD}$-DAG($\mathcal{A}$) implies Theorem 3.3 simply by combining the fact that computing the children of a node of $\mathcal{ACD}$-DAG($\mathcal{A}$) requires a time polynomial in $|\mathcal{Z}\text{-DAG}_{\mathcal{F}}| + |Q|$ (Lemma 3.6), and the fact that we call `ComputeChildren` exactly once per node of $\mathcal{ACD}$-DAG($\mathcal{A}$) in Algorithm 1 (as we never push back in nodesToTreat any set that was already explored, see line 7-8).

To establish Theorem 3.1, we remark that to compute $\mathcal{ACD}(\mathcal{A})$ from $\mathcal{Z}_{\mathcal{F}}$ and $\mathcal{A}$ we simply fold $\mathcal{Z}_{\mathcal{F}}$ to obtain $\mathcal{Z}\text{-DAG}_{\mathcal{F}}$, apply Theorem 3.3 to get $\mathcal{ACD}$-DAG($\mathcal{A}$), and then unfold the latter to obtain $\mathcal{ACD}(\mathcal{A})$. The first two steps requires a time polynomial in $|\mathcal{Z}\text{-DAG}_{\mathcal{F}}| + |Q| \leq |\mathcal{Z}_{\mathcal{F}}| + |Q|$, while the third step takes a time polynomial in $|\mathcal{ACD}(\mathcal{A})| \leq |Q| \cdot |\mathcal{Z}_{\mathcal{F}}|$.

Upper bound on the size of the ACD

▶ **Proposition 3.7.** *Let $\mathcal{A}$ be a Muller automaton and $\mathcal{F}$ the family defining its acceptance condition. Then,* $|\mathcal{ACD}(\mathcal{A})| \leq |Q| \cdot |\mathcal{Z}_{\mathcal{F}}|$.

We start by giving a technical lemma that will be useful for the subsequent analysis.

▶ **Lemma 3.8.** *Let $C \subseteq \Gamma$ and let n_C be a node in $\mathcal{Z}_{\mathcal{F}}$ such that $C \subseteq \nu(n_C)$. Let $D_1, \dots, D_k$ be k subsets of C such that, for all $i \neq j$, $C \in \mathcal{F} \iff D_i \notin \mathcal{F} \iff D_i \cup D_j \in \mathcal{F}$. Then, there are k strict descendants of n_C, $n_1, \dots, n_k$, such that $D_i \subseteq \nu(n_i)$, $\nu(n_i) \in \mathcal{F} \iff D_i \in \mathcal{F}$ and such that nodes n_i are pairwise incomparable for the ancestor relation. Moreover, these nodes can be computed in polynomial time in $|\mathcal{Z}_{\mathcal{F}}|$.*

Proof. To simplify notations we assume that $C \in \mathcal{F}$ and $D_i \notin \mathcal{F}$ (the proof is symmetric in the other case). For each D_i we pick a node n_i which is a descendant of n_C, such that $D_i \subseteq \nu(n_i)$ and maximal for $\preceq$ with this property. In particular, n_i is square and a strict descendant (Lemma 2.10). We prove that, for $j \neq i$, $D_j \nsubseteq \nu(n_i)$, implying that n_i and n_j are incomparable for the ancestor relation. Suppose by contradiction that for some $j \neq i$, $D_j \subseteq \nu(n_i)$. Then, $D_j \subseteq D_i \cup D_j \subseteq \nu(n_i)$, so, by Lemma 2.10, $D_i \cup D_j \notin \mathcal{F}$, contradicting the hypothesis. ◀

▶ **Lemma 3.9.** *For every state q, the tree $\mathcal{T}_q$ has size at most $|\mathcal{Z}_\mathcal{F}|$.*

Proof. 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. We let $C_n = \mathsf{col}(\nu(n))$ and $D_i = \mathsf{col}(\nu(n_i))$ be the colours labelling the cycles of these nodes. These sets satisfy that for $i \neq j$, $C_n \in \mathcal{F} \iff D_i \notin \mathcal{F} \iff D_i \cup D_j \in \mathcal{F}$. Indeed, if the union of D_i and D_j does not change the acceptance, we could take the union of the corresponding cycles, contradicting maximality. Lemma 3.8 provides k descendants of $f(n)$ such that the subtrees rooted at them are pairwise disjoint. This 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 $|Q| + |\mathcal{Z}_\mathcal{F}|$, concluding the proof of Proposition 3.7:

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

Upper bound on the size of the ACD-DAG

▶ **Proposition 3.10.** *Let $\mathcal{A}$ be a Muller automaton and $\mathcal{F}$ the family defining its acceptance condition. Then, $|\mathcal{ACD}\text{-}\mathsf{DAG}(\mathcal{A})| \leq |Q| \cdot |\mathcal{Z}\text{-}\mathsf{DAG}_\mathcal{F}|$.*

For obtaining this result, we want to follow the same proof scheme than in Proposition 3.7: our objective is to show that for all $q \in Q$, the local subDAG $\mathcal{D}_q$ can be embedded in $\mathcal{Z}\text{-}\mathsf{DAG}_\mathcal{F}$. However, we face a technical difficulty; in the case of the ACD we had that the subtrees rooted at k incomparable nodes were disjoint, which allowed us to carry out the recursion smoothly. This property no longer holds in DAGs.

▶ **Lemma 3.11.** *For every state q, the DAG $\mathcal{D}_q$ has size at most $|\mathcal{Z}\text{-}\mathsf{DAG}_\mathcal{F}|$.*

Proof. We will define an injective function $f\colon \mathcal{D}_q \to \mathcal{Z}\text{-}\mathsf{DAG}_\mathcal{F}$. For a node n in $\mathcal{D}_q$, we let $C_n = \mathsf{col}(\nu(n))$ be the set of colours appearing in the label of n. If n is not the root, we define $\mathsf{pred}(n)$ to be an immediate ancestor of n (that is, n is a child of $\mathsf{pred}(n)$). We let $\mathsf{pred}^*(n)$ be the sub-branch of nodes above n obtained by successive applications of pred, that is, $\mathsf{pred}^*(n) = \{n' \in \mathcal{D}_q \mid n' = \mathsf{pred}^k(n) \text{ for some } k\}$. We note that the elements of $\mathsf{pred}^*(n)$ are totally ordered by $\preceq$ (n being the maximal node and the root the minimal one).

We define f recursively: For the root n_0 of $\mathcal{D}_q$, we let $f(n_0)$ be a maximal node (for $\preceq$) in $\mathcal{Z}\text{-}\mathsf{DAG}_\mathcal{F}$ 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 (for $\preceq$) in the subDAG rooted at $f(\mathsf{pred}(n))$ containing C_n in its label. We remark that $f(n)$ is a round node if and only if n is a round node (by Lemma 2.10). Also, if n' is an ancestor of n in $\mathsf{pred}^*(n)$, then $f(n')$ is an ancestor of $f(n)$ in $\mathcal{Z}\text{-}\mathsf{DAG}_\mathcal{F}$.

We prove now the injectivity of f. Let n_1, n_2 be two different nodes in $\mathcal{D}_q$ (that is, $\nu(n_1) \neq \nu(n_2)$). First, we show that the colours appearing in their labels must differ.

▷ Claim 3.11.1. It is satisfied that $C_{n_1} \neq C_{n_2}$.

Proof. Suppose by contradiction that $C_{n_1} = C_{n_2}$. Then, any node n containing $\nu(n_1)$ in its label satisfies that $\nu(n)$ is an accepting cycle if and only if $\nu(n) \cup \nu(n_2)$ is an accepting cycle. Let n be a node of minimal depth such that $\nu(n_1) \subseteq \nu(n)$ and $\nu(n_2) \nsubseteq \nu(n)$. The label of an immediate predecessor of n contains $\nu(n_1) \cup \nu(n_2)$ by minimality. This leads to a contradiction, as $\nu(n) \subsetneq \nu(n) \cup \nu(n_2)$, so $\nu(n)$ would not be a maximal subcycle producing an alternation in the acceptance status. ◁

We assume w.l.o.g. that n_1 is round (that is, $C_{n_1} \in \mathcal{F}$). Suppose by contradiction that $f(n_1) = f(n_2)$. Then, n_2 is also round, and it is satisfied that $C_{n_1} \cup C_{n_2} \subseteq f(n_1)$, by definition of f. Again by definition of f, no child of $f(n_1)$ contains $C_1 \cup C_2$, so, by Lemma 2.10, $C_1 \cup C_2 \in \mathcal{F}$. Let n' be the minimal node in $\mathsf{pred}^*(n_1)$ such that $\nu(n_2) \subseteq \nu(n')$. We do the prove for the case in which n' is round, the other case is symmetric. Let $\tilde{n}$ be the child of n' in $\mathsf{pred}^*(n')$, which is a square node. We claim that the following three properties hold:

i) $C_{\tilde{n}} \cup C_{n_2} \in \mathcal{F}$,
ii) $C_{\tilde{n}} \cup C_{n_2} \subseteq f(\tilde{n})$, and
iii) no child of $f(\tilde{n})$ contains $C_{\tilde{n}} \cup C_{n_2}$.

This leads to a contradiction, as the second and third items, combined with Lemma 2.10 and the fact that $f(\tilde{n})$ is a square node, imply that $C_{\tilde{n}} \cup C_{n_2} \notin \mathcal{F}$. We prove the three items:

i) Follows from the fact that $\nu(\tilde{n})$ is a maximal rejecting cycle of $\nu(n')$, but $\nu(n')$ contains $\nu(\tilde{n}) \cup \nu(n_2)$.
ii) By definition of f, $C_{\tilde{n}} \subseteq f(\tilde{n})$. Also, the node $f(\tilde{n})$ is an ancestor of $f(n_2)$, so $C_{n_2} \subseteq f(n_2) \subseteq f(\tilde{n})$.
iii) By definition of f, no child of $f(\tilde{n})$ contains $C_{\tilde{n}}$ in its label. ◀

We conclude that the size of $\mathcal{ACD}\text{-}\mathsf{DAG}(\mathcal{A})$ is polynomial in $|Q| + |\mathcal{Z}\text{-}\mathsf{DAG}_{\mathcal{F}}|$:

$$|\mathcal{ACD}\text{-}\mathsf{DAG}(\mathcal{A})| \leq \sum_{q \in Q} |\mathcal{D}_q| \leq |Q| \cdot |\mathcal{Z}\text{-}\mathsf{DAG}_{\mathcal{F}}|.$$

4 Minimisation of colours and Rabin pairs

We consider the problem of minimising the representation of the acceptance condition of automata. That is, given an 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 question of 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 4.2 and 4.5). An alternative point of view over the minimisation of Rabin pairs, using generalised Horn formulas, is presented in Appendix A. Then, we consider the question taking into account the structure of the automaton. Surprisingly, we show that in this case both problems are NP-hard, even if the ACD is given as input (Theorems 4.13 and 4.15).

4.1 Minimisation of the representation of Muller languages in polynomial time

4.1.1 Minimisation of colours for Muller languages

We say that a Muller language $L \subseteq \Sigma^\omega$ is *k-colour type* if there is a set of k colours Γ, a Muller language $L' \subseteq \Gamma^\omega$ and a mapping $\phi\colon \Sigma \to \Gamma$ such that for all $w \in \Sigma^\omega$, $w \in L \iff \phi(w) \in L'$.

▶ Remark 4.1. A Muller language $L \subseteq \Sigma^\omega$ is k-colour type if and only if it can be recognised by a deterministic Muller automaton with one state using k output colours.

Also, L is k-colour type if and only if all automata $\mathcal{A}$ using L as acceptance condition can be relabelled with an equivalent Muller condition using at most k colours.

Problem:	Colour-Minimisation-ML
Input:	A Muller language $\mathsf{Muller}_\Sigma(\mathcal{F})$ represented by the Zielonka DAG $\mathcal{Z}\text{-}\mathsf{DAG}_\mathcal{F}$ and a positive integer k.
Question:	Is $\mathsf{Muller}_\Sigma(\mathcal{F})$ k-colour type?

We could have chosen other representations of $\mathsf{Muller}_\Sigma(\mathcal{F})$ for the input of this problem (mainly, colour-explicitly or as a Zielonka tree). We have chosen to specify the input as a Zielonka DAG, as it is more succinct than the other representations (c.f. Figure 3 and Propositions 5.6, 5.7). We now prove that this problem can be solved in polynomial time if $\mathcal{F}$ is represented as a Zielonka DAG, which implies that it can be equally solved in polynomial time if $\mathcal{F}$ is represented colour-explicitly or as a Zielonka tree.

▶ **Theorem 4.2** (Tractability of minimisation of colours for Muller languages). *The problem* Colour-Minimisation-ML *can be solved in polynomial time.*

Proof. We say that two letters $a, b \in \Sigma$ are *$\mathcal{F}$-equivalent*, written $a \sim_\mathcal{F} b$, if, for all $C \subseteq \Sigma$:

$$C \cup \{a\} \in \mathcal{F} \iff C \cup \{b\} \in \mathcal{F} \iff C \cup \{a, b\} \in \mathcal{F}.$$

It is immediate to check that $\sim_\mathcal{F}$ is indeed an equivalence relation. We let $[a]$ denote the equivalence class of a for $\sim_\mathcal{F}$, $\Sigma/_{\sim_\mathcal{F}}$ the set of equivalence classes, and for $C \subseteq \Sigma$, we write $\pi(C) = \{[a] \mid a \in C\}$.

▷ Claim 4.2.1. For all $C \subseteq \Sigma$, $C \in \mathcal{F} \iff \bigcup_{a\in C}[a] \in \mathcal{F}$.

Proof. For each $a \in C$, we can add all the elements in $[a]$ one by one to C without changing the acceptance status. ◁

▷ Claim 4.2.2. $\mathsf{Muller}_\Sigma(\mathcal{F})$ is k-colour type if and only if there are at most k classes for the $\mathcal{F}$-equivalence relation.

Proof. Assume that $\mathsf{Muller}_\Sigma(\mathcal{F})$ is k-colour type. Then, there is a Muller automaton $\mathcal{A}$ with one state q and using a Muller acceptance condition over Γ, with $|\Gamma| = k$. This defines a function $f\colon \Sigma \to \Gamma$ that sends a to α if α is the colour produced when reading a, that is, if $q \xrightarrow{a:\alpha} q$ is the corresponding transition in $\mathcal{A}$. It is immediate that if $f(a) = f(b)$, then $a \sim_\mathcal{F} b$, so there are no more than $|\Gamma| = k$ equivalence classes for $\sim_\mathcal{F}$.

Conversely, we can define a one-state Muller automaton using as output colours the equivalence classes for $\sim_\mathcal{F}$: $q \xrightarrow{a:[a]} q$, using as acceptance condition the language associated to:

$$\widetilde{\mathcal{F}} = \{\pi(C) \mid C \in \mathcal{F}\}.$$

The fact that the obtained automaton recognises $\mathsf{Muller}_\Sigma(\mathcal{F})$ follows from Claim 4.2.1. ◁

▷ Claim 4.2.3. Two letters $a, b \in \Sigma$ are $\mathcal{F}$-equivalent if and only for every node n of the Zielonka DAG of $\mathcal{F}$, $a \in \nu(n) \iff b \in \nu(n)$.

Proof. Suppose that there is a node n in the Zielonka DAG such that $a \in \nu(n)$ and $b \notin \nu(n)$. Take a minimal node with this property, and let n' be its parent. Then, by minimality, $a \in \nu(n)$, $b \notin \nu(n)$, but $a, b \in \nu(n')$. By Remark 2.9, we obtain that $\nu(n) = \nu(n) \cup \{a\} \in \mathcal{F} \iff \nu(n) \cup \{a, b\} \notin \mathcal{F}$, so a and b are not $\mathcal{F}$-equivalent.

For the other direction, let $C \subseteq \Sigma$ and take a minimal node n such that $C \cup \{a\} \subseteq \nu(n)$. Then, $b \in \nu(n)$, so n is also minimal such that $C \cup \{a, b\} \subseteq \nu(n)$, and therefore $C \cup \{a\} \in \mathcal{F} \iff C \cup \{a, b\} \in \mathcal{F}$. The third equivalence is obtained by symmetry. ◁

Claim 4.2.2 tells us that in order to minimise the number of required colours, we need to compute the classes of the $\mathcal{F}$-equivalence relation. This can be directly done by inspecting the Zielonka DAG by Claim 4.2.3. ◀

4.1.2 Minimisation of Rabin pairs for Rabin languages

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 $w \in \Sigma^\omega$, $w \in L \iff \phi(w) \in \mathsf{Rabin}_\Gamma(\mathcal{R})$.

▶ Remark 4.3. A Rabin language $L \subseteq \Sigma^\omega$ is k-Rabin-pair type if and only if it can be recognised by a deterministic Rabin automaton with one state using k Rabin pairs.

▶ Remark 4.4. If $L \subseteq \Sigma^\omega$ is k-Rabin-pair type, then there exists a family of k Rabin pairs $\mathcal{R}'$ over Σ such that $L = \mathsf{Rabin}_\Sigma(\mathcal{R}')$.

Proof. Let $\mathcal{R} = \{(\mathfrak{g}_1, \mathfrak{r}_1), \dots, (\mathfrak{g}_k, \mathfrak{r}_k)\}$ be a set of Rabin pairs over Γ and let $\phi\colon \Sigma \to \Gamma$ such that for all $w \in \Sigma^\omega$, $w \in L \iff \phi(w) \in \mathsf{Rabin}_\Gamma(\mathcal{R})$. It suffices to define $\mathcal{R}' = \{(\mathfrak{g}'_1, \mathfrak{r}'_1), \dots, (\mathfrak{g}'_k, \mathfrak{r}'_k)\}$ with $\mathfrak{g}'_i = \phi^{-1}(\mathfrak{g}_i)$ and $\mathfrak{r}'_i = \phi^{-1}(\mathfrak{g}_i)$. ◀

Problem:	RABIN-PAIR-MINIMISATION-ML
Input:	A family of Rabin pairs $\mathcal{R}$ over Σ and a positive integer k.
Question:	Is $\mathsf{Rabin}_\Sigma(\mathcal{R})$ k-Rabin-pair type?

▶ **Theorem 4.5** (Tractability of minimisation of Rabin pairs for Rabin languages)**.** *The problem* RABIN-PAIR-MINIMISATION-ML *can be solved in polynomial time.*

Given a set of Rabin pairs $\mathcal{R} = \{(\mathfrak{g}_1, \mathfrak{r}_1), \dots, (\mathfrak{g}_r, \mathfrak{r}_r)\}$ over Σ and a set $S \subseteq \Sigma$, we say that S *satisfies* (or that is *accepted by*) $\mathcal{R}$ if, for some i, $S \cap \mathfrak{g}_i \neq \emptyset$ and $S \cap \mathfrak{r}_i = \emptyset$. Otherwise, we say that S is rejected by $\mathcal{R}$. By a small abuse of notation, we write $S \in \mathsf{Rabin}_\Sigma(\mathcal{R})$ (resp. $S \notin \mathsf{Rabin}_\Sigma(\mathcal{R})$) if S is accepted by (resp. rejected by) $\mathcal{R}$. We define the same notions for Streett conditions symmetrically.

▶ **Lemma 4.6.** *Let $\mathcal{R}$ be a family of Rabin pairs over Σ and let $S \subseteq \Sigma$. There exists a maximum subset of S rejected by $\mathcal{R}$, and it is computable in polynomial time.*

Proof. We describe an algorithm building a decreasing sequences of subsets of S. Initially, set $T = S$. While there exists $(\mathfrak{g}, \mathfrak{r}) \in \mathcal{R}$ satisfied by T, set $T = T \setminus \mathfrak{g}$. This algorithm maintains the invariant that all sets rejected by $\mathcal{R}$ should be included in T. Furthermore, it terminates in at most $|\Sigma|$ iterations as T strictly decreases at each step. In the end, we obtain a set that does not satisfy $\mathcal{R}$, and that is maximum by the invariant property. ◀

▶ **Lemma 4.7.** *Given two families $\mathcal{R}, \mathcal{R}'$ of Rabin pairs over Σ, there is a polynomial-time algorithm that checks whether $\mathsf{Rabin}_\Sigma(\mathcal{R}) \nsubseteq \mathsf{Rabin}_\Sigma(\mathcal{R}')$ and returns a maximal set $S \in \mathsf{Rabin}_\Sigma(\mathcal{R}) \setminus \mathsf{Rabin}_\Sigma(\mathcal{R}')$ if it is the case.*

Proof. For each $(\mathfrak{g}, \mathfrak{r}) \in \mathcal{R}$, we apply the following procedure.

Set $S = \Sigma \setminus \mathfrak{r}$, and define $S_{(\mathfrak{g},\mathfrak{r})}$ as the maximum subset of S not satisfying $\mathcal{R}'$. We can compute $S_{(\mathfrak{g},\mathfrak{r})}$ by Lemma 4.6. If $S_{(\mathfrak{g},\mathfrak{r})} \cap \mathfrak{g} = \emptyset$, then it does not satisfy $(\mathfrak{g}, \mathfrak{r})$, hence $\mathsf{Rabin}_\Sigma((\mathfrak{g}, \mathfrak{r})) \setminus \mathsf{Rabin}_\Sigma(\mathcal{R}') = \emptyset$. If this is the case for all $(\mathfrak{g}, \mathfrak{r})$, then we can conclude that $\mathsf{Rabin}_\Sigma(\mathcal{R}) \subseteq \mathsf{Rabin}_\Sigma(\mathcal{R}')$. Otherwise, we can select a maximal set among the $S_{(\mathfrak{g},\mathfrak{r})}$ such that $S_{(\mathfrak{g},\mathfrak{r})} \cap \mathfrak{g} \neq \emptyset$.

This yields a maximal set in $\mathsf{Rabin}_\Sigma(\mathcal{R}) \setminus \mathsf{Rabin}_\Sigma(\mathcal{R}')$. ◀

In Algorithm 3 we give a procedure minimising the number of Rabin pairs. We remark that the Rabin condition built by this algorithm uses the same set of colours as the input Rabin condition.

Algorithm 3 Minimisation algorithm for Rabin conditions.

Input: A set of Rabin pairs $\mathcal{R}$ over Σ
$\mathcal{R}_{\min} \leftarrow \{\}$
while $\mathsf{Rabin}_\Sigma(\mathcal{R}) \nsubseteq \mathsf{Rabin}_\Sigma(\mathcal{R}_{\min})$ **do**
  $S \leftarrow$ maximal set in $\mathsf{Rabin}_\Sigma(\mathcal{R}) \setminus \mathsf{Rabin}_\Sigma(\mathcal{R}_{\min})$
  $T \leftarrow$ maximum subset of S not in $\mathsf{Rabin}_\Sigma(\mathcal{R})$
  $\mathcal{R}_{\min} \leftarrow \mathcal{R}_{\min} \cup \{(\Sigma \setminus T, \Sigma \setminus S)\}$
end while
return $\mathcal{R}_{\min}$

▶ **Lemma 4.8.** *Algorithm 3 terminates and $\mathsf{Rabin}_\Sigma(\mathcal{R}_{\min}) = \mathsf{Rabin}_\Sigma(\mathcal{R})$.*

Proof. The algorithm clearly terminates, as $\mathsf{Rabin}_\Sigma(\mathcal{R}_{\min})$ increases at each iteration of the loop. Thus, we eventually get out of the loop, hence $\mathsf{Rabin}_\Sigma(\mathcal{R}) \subseteq \mathsf{Rabin}_\Sigma(\mathcal{R}_{\min})$. Furthermore $\mathsf{Rabin}_\Sigma(\mathcal{R}_{\min}) \subseteq \mathsf{Rabin}_\Sigma(\mathcal{R})$ is a loop invariant: at the start we have $\mathsf{Rabin}_\Sigma(\mathcal{R}_{\min}) = \emptyset$, and at each loop iteration we add to $\mathcal{R}_{\min}$ a pair $(\Sigma \setminus T, \Sigma \setminus S)$ such that $S \in \mathsf{Rabin}_\Sigma(\mathcal{R})$ and T is the maximum subset of S rejected by $\mathcal{R}$. As a consequence, since $\mathsf{Rabin}_\Sigma((\Sigma \setminus T, \Sigma \setminus S))$ contains only sets included in S but not in T, $\mathsf{Rabin}_\Sigma((\Sigma \setminus T, \Sigma \setminus S)) \subseteq \mathsf{Rabin}_\Sigma(\mathcal{R})$ and thus the invariant is maintained. ◀

▶ **Lemma 4.9.** *Let $\mathcal{R}_{\min}$ be the set of Rabin pairs obtained by applying Algorithm 3 on a set of Rabin pairs $\mathcal{R}$. Let $(\mathfrak{g}_1, \mathfrak{r}_1) \neq (\mathfrak{g}_2, \mathfrak{r}_2) \in \mathcal{R}_{\min}$, then $\Sigma \setminus \mathfrak{r}_1 \cup \Sigma \setminus \mathfrak{r}_2$ is not accepted by $\mathcal{R}_{\min}$.*

Proof. Suppose by contradiction that $\Sigma \setminus \mathfrak{r}_1 \cup \Sigma \setminus \mathfrak{r}_2$ is accepted by $\mathcal{R}_{\min}$. Then there exists $(\mathfrak{g}_3, \mathfrak{r}_3) \in \mathcal{R}_{\min}$ accepting $\Sigma \setminus \mathfrak{r}_1 \cup \Sigma \setminus \mathfrak{r}_2$. As a consequence, neither $\Sigma \setminus \mathfrak{r}_1$ nor $\Sigma \setminus \mathfrak{r}_2$ intersects $\mathfrak{r}_3$, and one of them intersects $\mathfrak{g}_3$. Therefore, $(\mathfrak{g}_3, \mathfrak{r}_3)$ necessarily accepts either $\Sigma \setminus \mathfrak{r}_1$ or $\Sigma \setminus \mathfrak{r}_2$. Without loss of generality, we assume that it accepts $\Sigma \setminus \mathfrak{r}_1$. In particular, we have $\mathfrak{r}_3 \subseteq \mathfrak{r}_1$.

During the execution of Algorithm 3 on $\mathcal{R}$ resulting in $\mathcal{R}_{\min}$, S must have taken the values $\Sigma \setminus \mathfrak{r}_j$ for both $j = 1$ and $j = 3$, starting with $j = 3$ since S is always taken maximal and $\Sigma \setminus \mathfrak{r}_1 \subseteq \Sigma \setminus \mathfrak{r}_3$. However, after adding $(\mathfrak{g}_3, \mathfrak{r}_3)$ to $\mathcal{R}_{\min}$, $\Sigma \setminus \mathfrak{r}_1$ is accepted by $\mathcal{R}_{\min}$, contradicting the fact that $(\mathfrak{g}_1, \mathfrak{r}_1)$ is in $\mathcal{R}_{\min}$ in the end. ◀

By Remark 4.4, it suffices to check minimality of $\mathcal{R}_{\min}$ amongst families of Rabin pairs over the alphabet Σ.

▶ **Lemma 4.10.** *Let $\mathcal{R}$ and $\tilde{\mathcal{R}}$ be two families of Rabin pairs over Σ such that $\mathsf{Rabin}_\Sigma(\mathcal{R}) = \mathsf{Rabin}_\Sigma(\tilde{\mathcal{R}})$, and let $\mathcal{R}_{\min}$ the family returned by Algorithm 3 when applied on $\mathcal{R}$. Then, $|\mathcal{R}_{\min}| \leq |\tilde{\mathcal{R}}|$.*

Proof. In order to prove the lemma, we map each pair $(\mathfrak{g}, \mathfrak{r})$ of $\mathcal{R}_{\min}$ to a pair $(\tilde{\mathfrak{g}}, \tilde{\mathfrak{r}})$ of $\tilde{\mathcal{R}}$ and prove that the mapping is injective.

By Lemma 4.8, we have $\mathsf{Rabin}_\Sigma(\mathcal{R}) = \mathsf{Rabin}_\Sigma(\mathcal{R}_{\min})$, and thus $\mathsf{Rabin}_\Sigma(\tilde{\mathcal{R}}) = \mathsf{Rabin}_\Sigma(\mathcal{R}_{\min})$. For all $(\mathfrak{g}, \mathfrak{r}) \in \mathcal{R}_{\min}$, as $\Sigma \setminus \mathfrak{r}$ is accepted by $\mathcal{R}_{\min}$ and $\mathsf{Rabin}_\Sigma(\tilde{\mathcal{R}}) = \mathsf{Rabin}_\Sigma(\mathcal{R}_{\min})$, we can find $(\tilde{\mathfrak{g}}, \tilde{\mathfrak{r}}) \in \tilde{\mathcal{R}}$ such that $\Sigma \setminus \mathfrak{r}$ intersects $\tilde{\mathfrak{g}}$ but not $\tilde{\mathfrak{r}}$.

Now assume that there exist $(\mathfrak{g}_1, \mathfrak{r}_1) \neq (\mathfrak{g}_2, \mathfrak{r}_2) \in \mathcal{R}_{\min}$ such that $(\tilde{\mathfrak{g}}_1, \tilde{\mathfrak{r}}_1) = (\tilde{\mathfrak{g}}_2, \tilde{\mathfrak{r}}_2)$. The pair $(\tilde{\mathfrak{g}}_1, \tilde{\mathfrak{r}}_1)$ then accepts both $\Sigma \setminus \mathfrak{r}_1$ and $\Sigma \setminus \mathfrak{r}_2$. As a consequence, both $\Sigma \setminus \mathfrak{r}_1$ and $\Sigma \setminus \mathfrak{r}_2$ intersect $\tilde{\mathfrak{g}}_1$ and neither intersects $\tilde{\mathfrak{r}}_1$, thus $\Sigma \setminus \mathfrak{r}_1 \cup \Sigma \setminus \mathfrak{r}_2$ is also accepted by $(\tilde{\mathfrak{g}}_1, \tilde{\mathfrak{r}}_1)$. This contradicts Lemma 4.9.

As a consequence, for all $(\mathfrak{g}_1, \mathfrak{r}_1), (\mathfrak{g}_2, \mathfrak{r}_2) \in \mathcal{R}_{\min}$, $(\tilde{\mathfrak{g}}_1, \tilde{\mathfrak{r}}_1) \neq (\tilde{\mathfrak{g}}_2, \tilde{\mathfrak{r}}_2)$. As a result, $\tilde{\mathcal{R}}$ must contain at least as many pairs than $\mathcal{R}_{\min}$, proving the lemma. ◀

▶ **Proposition 4.11.** *Algorithm 3 terminates in polynomial time and returns a family of Rabin pairs with the same Rabin language as the input and with a minimal number of pairs.*

Proof. By Lemmas 4.8 and 4.10, the algorithm terminates and returns a family of Rabin pairs with the desired property. Furthermore, since a pair is added to $\mathcal{R}_{\min}$ at each iteration of the loop, and since the resulting family contains at most $|\mathcal{R}|$ pairs (by minimality), the algorithm terminates after at most $|\mathcal{R}|$ iterations. Finally, by Lemmas 4.6 and 4.7, each iteration can be done in polynomial time, hence the algorithm terminates in polynomial time. ◀

▶ **Corollary 4.12.** *Given a set of Rabin pairs $\mathcal{R}$, one can compute a set $\mathcal{R}_{min}$ with the same Streett language and a minimal number of clauses.*

Proof. It suffices to observe that two sets of Rabin pairs $\mathcal{R}_1, \mathcal{R}_2$ have the same Rabin language if and only if they have the same Streett language. Hence by Proposition 4.11, Algorithm 3 also minimises the number of pairs for the Streett language. ◀

4.2 Minimisation of acceptance conditions on top of an automaton is NP-hard

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 the problems become NP-hard when taking into account the structure of the automata.

4.2.1 Minimisation of colours on top of a Muller automaton

We say that a 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 $\mathcal{A}$.

Problem:	COLOUR-MINIMISATION-AUT
Input:	A Muller automaton $\mathcal{A}$ and a positive integer k.
Question:	Is $\mathcal{A}$ k-colour type?

We remark that we have not specified the representation of the acceptance condition of $\mathcal{A}$, therefore, this problem admits different variants according to this representation. We will show that for the three representations we are concerned with (colour-explicit, Zielonka tree and Zielonka DAG), the problem COLOUR-MINIMISATION-AUT is NP-hard. This implies that the problem is NP-hard even if the ACD is provided as input, by Theorem 3.1.

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 [22]. However, his reduction does not directly generalise to *transition-based automata*; the reduction we propose is of a quite different nature. Moreover, our reduction of NP-hardness uses an automaton with only 2 disconnected states. This is quite surprising, as we could think that a generalisation of the methods from the proof of Theorem 4.2 to the ACD would yield a polynomial time algorithm. However, it is not possible to properly combine the structure of the local subtrees of the ACD.

▶ **Theorem 4.13** (Hardness of minimisation of colours for Muller automata). *The problem* COLOUR-MINIMISATION-AUT *is* NP*-hard, if the acceptance condition* $\mathsf{Muller}_\Gamma(\mathcal{F})$ *of* $\mathcal{A}$ *is represented colour-explicitly, as a Zielonka tree or as the ACD of* $\mathcal{A}$*.*

We prove the result for the representations colour-explicit and Zielonka tree. The result for the other representations follows then from Proposition 5.6 and Theorem 3.1.

We give a reduction from the problem MAX-CLIQUE defined as follows. An *(undirected) graph* is a pair $G = (V, E)$ consisting of a set of vertices V and a set of edges $E \subseteq \binom{V}{2}$ (that is, edges are subsets of size exactly two, in particular, no self loops are allowed). We say that a graph is *connected* if there is a path connecting any pair of vertices. A *clique* of G is a subset $V' \subseteq V$ such that $\{v', u'\} \in E$ for every $v' \neq u' \in V'$. The problem MAX-CLIQUE consists in, given a graph G (that can be assumed connected) and a positive integer k, decide whether G contains a clique of size k. The problem MAX-CLIQUE is well-known to be NP-complete [26].

Let $G = (V, E)$ be a simple, connected undirected graph and $k \in \mathbb{N}$. We consider the automaton $\mathcal{A}_{G,k}$ defined as:

- It has two states q_{vert} (which is initial) and q_k.
- The input alphabet is $\Sigma = V \cup A_k \cup \{x\}$, where A_k is a set of size k disjoint from V and x is a fresh letter.

[5] As of today, the proof is not currently publicly available online, we got access to it by a personal communication. The statement of the theorem only express 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.

- The set of output colours is $\Gamma = V \cup A_k$.
- The transitions of $\mathcal{A}_{G,k}$ are given by:
  - $q_{\mathsf{vert}} \xrightarrow{v:v} q_{\mathsf{vert}}$ for every $v \in V$,
  - $q_{\mathsf{vert}} \xrightarrow{x:y} q_k$ (where $y \in \Gamma$ is irrelevant), and
  - $q_k \xrightarrow{a:a} q_k$ for every $a \in A_k$.
- Its acceptance condition is the Muller language associated to the family:

$$\mathcal{F} = E \cup \{\{a, a'\} \mid a, a' \in A_k,\ a \neq a'\}.$$

The representation of this automaton is polynomial in $|G| + k$, since $|\mathcal{F}| = \mathcal{O}(|E| + k^2)$. We also note that the Zielonka tree of $\mathcal{F}$ has size $\mathcal{O}(|E| + k^2)$.

We will use the following property satisfied by $\mathcal{L}(\mathcal{A}_{G,k})$:

- For all $\alpha \in \Sigma$, words ending by α^ω are not in $\mathcal{L}(\mathcal{A}_{G,k})$ (cycles consisting in a single self loop are rejecting).
- For all $a, b \in A_k$, $a \neq b$, $x(ab)^\omega \in \mathcal{L}(\mathcal{A}_{G,k})$.

▶ **Lemma 4.14.** *G admits a clique of size k if and only if $\mathcal{A}_{G,k}$ is $|V|$-colour type.*

Proof. Assume that $V' = \{v'_1, \dots, v'_k\}$ is a clique of size k of G, and let $A_k = \{a_1, \dots, a_k\}$. We consider the Muller condition using as set of colours $\Gamma' = V$ and given by $\mathcal{F}' = E$. The new acceptance condition over $\mathcal{A}_{G,k}$ is obtained by using the same colouring for the self loops over q_{vert}, and recolouring self loops $q_k \xrightarrow{a_i:a_i} q_k$ with $q_k \xrightarrow{a_i:v'_i} q_k$. It is immediate that the obtained acceptance condition is equivalent to the original one of $\mathcal{A}_{G,k}$.

For the converse, assume that $\mathcal{A}_{G,k}$ is $|V|$-colour type. Then there is a set Γ' of $|V|$ colours and a colouring function $\mathsf{col}' \colon \Delta \to \Gamma'$ yielding an equivalent condition over $\mathcal{A}_{G,k}$.

First, we show that for two different self loops $e_1 = q_{\mathsf{vert}} \xrightarrow{v_1:c_1} q_{\mathsf{vert}}$ and $e_2 = q_{\mathsf{vert}} \xrightarrow{v_2:c_2} q_{\mathsf{vert}}$, we have $c_1 \neq c_2$ (where $c_1 = \mathsf{col}'(e_1)$ and $c_2 = \mathsf{col}'(e_2)$). If $\{v_1, v_2\} \in E$, this is clear, as $\{e_1\}$ is a rejecting cycle, but $\{e_1, e_2\}$ is accepting. Suppose that $\{v_1, v_2\} \notin E$, and let $u \in V$ such that $\{v_1, u\} \in E$ (which exists as G is connected). Then, the cycle $\{e_1, q_{\mathsf{vert}} \xrightarrow{u} q_{\mathsf{vert}}\}$ is accepting while $\{e_1, e_2, q_{\mathsf{vert}} \xrightarrow{u} q_{\mathsf{vert}}\}$ is rejecting, so they cannot be coloured equally. Therefore, for each colour $c \in \Gamma'$ there is one self loop v such that $\mathsf{col}'(v) = c$.

Secondly, we remark that for two different self loops $e_1 = q_k \xrightarrow{a_1:c_1} q_k$ and $e_2 = q_k \xrightarrow{a_2:c_2} q_k$ over q_k it is also satisfied that $c_1 = \mathsf{col}'(e_1) \neq c_2 = \mathsf{col}'(e_2)$, as $xa_1^\omega \notin \mathcal{L}(\mathcal{A}_{G,k})$, but $x(a_1a_2)^\omega \in \mathcal{L}(\mathcal{A}_{G,k})$. Let $\{c_1, \dots, c_k\}$ be the k different colours labelling the self loops over q_k. We obtain that the subset $\{v_1, \dots, v_k\} \subseteq V$ of vertices such $\mathsf{col}'(v_i) = c_i$ form a clique of size k in G. ◀

4.2.2 Minimisation of Rabin pairs on top of a Rabin automaton

Similarly, we consider the problem of minimising the number Rabin pairs over a fixed Rabin automaton.

We say that a 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.

Problem:	RABIN-PAIR-MINIMISATION-AUT
Input:	A Rabin automaton $\mathcal{A}$ and a positive integer k.
Question:	Is $\mathcal{A}$ k-Rabin-pair type?

As before, we can consider different representations of the acceptance condition $\mathsf{Rabin}_\Gamma(\mathcal{R})$ of the automaton: using Rabin pairs, with a colour-explicit Muller condition, or by providing the Zielonka tree, the Zielonka DAG or the ACD.

▶ **Theorem 4.15** (Hardness 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.*

▶ **Lemma 4.16.** *The problem* Rabin-Pair-Minimisation-Aut *is in* NP*.*

Proof. It suffices to guess a family of k Rabin pairs over the set of colours Δ and check if the obtained automaton recognises the same language as before. For the representation as Rabin pairs, this can be done in polynomial time as the equivalence of deterministic Rabin automata can be checked in polynomial time [13]. Propositions 5.6 and 5.8 imply that this is also possible for the other representations. ◀

For the NP-hardness part, we reduce from the problem Chromatic number, defined as follows. A *k-colouring* of an undirected graph $G = (V, E)$ is a mapping $\chi : V \to \{1, \dots, k\}$ such that $\chi(v) = \chi(v') \Rightarrow \{v, v'\} \notin E$ for every pair of nodes $v, v' \in V$. The problem Chromatic number consists in, given a graph G (that can be assumed connected) and a positive integer k, decide whether G admits a k-colouring. The problem Chromatic number is well-known to be NP-complete [26].

Let $G = (V, E)$ be a connected graph. We select an arbitrary vertex $v_{\mathsf{init}} \in V$. A *pseudo-path* in G is a (finite or infinite) sequence $v_0 e_0 v_1 e_1 \cdots \in (V \cup E)^\infty$ such that $v_i, v_{i+1} \in e_i$ for all i. Note that we allow v_i and v_{i+1} to be equal, hence the term pseudo-path; that is, we allow a pseudo-path to step on an edge without going through it, and come back to the previous vertex. A pseudo-path is *initial* if $v_0 = v_{\mathsf{init}}$. We say that such a pseudo-path *stabilises around* v if it is infinite and there exists i such that for all $j > i$, $v_j = v$, i.e., the pseudo-path eventually stays on the same vertex and just steps on the adjacent edges. We write $\mathsf{Stab}(v)$ for the set of initial pseudo-paths stabilising around v. For $v \in V$, we write $\mathsf{adj}(v)$ for the set of edges $\{e \in E \mid v \in e\}$.

We define the automaton $\mathcal{A}_G$ as follows:

- $Q = V \cup E \cup \{q_{\mathsf{init}}\}$, where q_{init} is a fresh element, which is the initial state,
- $\Sigma = \Gamma = V \cup E$,
- $\Delta = \{(q_{\mathsf{init}}, v_{\mathsf{init}}, v_{\mathsf{init}})\} \cup \{(v, e, e) \in V \times E^2 \mid v \in e\} \cup \{(e, v, v) \in E \times V^2 \mid v \in e\}$,
- the colour of each transition is the letter it reads, $\mathsf{col}(q \xrightarrow{x} q') = x$,
- $W = \mathsf{Rabin}_{V \cup E}(\mathcal{R})$ with $\mathcal{R} = \{\big(V \cup E,\ (V \cup E) \setminus (\{v\} \cup \mathsf{adj}(v))\big) \mid v \in V\}$.

This automaton is deterministic and recognises the language $\bigcup_{v \in V} \mathsf{Stab}(v)$. Note that $\mathcal{A}$ is not complete: it only reads initial pseudo-paths of G.

The representation of the automaton $\mathcal{A}_G$ is polynomial in $|V| + |E|$. Moreover, we note that the Zielonka tree of $\mathsf{Rabin}_{V \cup E}(\mathcal{R})$ has size at most $2|V| + 1$, so by Theorem 3.1, we can provide in polynomial time $\mathcal{ACD}(\mathcal{A}_G)$. To obtain the NP-hardness when the acceptance condition is represented colour-explicitly, we note that the problem Chromatic number remains NP-hard over graphs of outdegree at most 4 [19]. For those graphs, a family $\mathcal{F}$ representing $\mathsf{Rabin}_{V \cup E}(\mathcal{R})$ is of polynomial size.

▶ **Lemma 4.17.** *A connected graph G admits a k-colouring if and only if $\mathcal{A}_G$ is k-Rabin-pair type.*

Proof. Let $\chi\colon V \to \{1,\dots,k\}$ be a k-colouring of G. For all $i \in \{1,\dots,k\}$ we define the Rabin pair $R_i = (\mathfrak{g}_i, \mathfrak{r}_i)$ with:

$$\mathfrak{g}_i = V \cup E\,, \quad \mathfrak{r}_i = (V \cup E) \setminus \bigcup_{v \in \chi^{-1}(i)} \{v\} \cup \mathsf{adj}(v).$$

We set $\mathcal{R}' = \{R_i \mid i \in \{1,\dots,k\}\}$. Let $\mathcal{A}'$ be the automaton obtained by setting the acceptance condition of $\mathcal{A}_G$ to be $\mathsf{Rabin}_{V\cup E}(\mathcal{R}')$. Let us prove that $\mathcal{L}(\mathcal{A}') = \mathcal{L}(\mathcal{A}_G)$. Let $w \in \mathcal{L}(\mathcal{A})$, w is an initial pseudo-path and there must exist v such that w stabilises around v. Let $i = \chi(v)$, ultimately w only visits $v \cup \mathsf{adj}(v)$ and thus it satisfies R_i. As a result, $w \in \mathcal{L}(\mathcal{A}')$. Let $w \in \mathcal{L}(\mathcal{A}')$. Again, w is an initial pseudo-path and there must exist i such that w ultimately only visits $\bigcup_{v\in\chi^{-1}(i)}\{v\} \cup \mathsf{adj}(v)$. Moreover, as χ is a k-colouring of G, $\chi^{-1}(i)$ is an independent set, hence w cannot visit infinitely often two distinct vertices from this set without visiting infinitely often an intermediate vertex of a different colour. As a consequence, w must stabilise around some $v \in \chi^{-1}(i)$, thus $w \in \mathcal{L}(\mathcal{A})$. We have shown that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$.

For the other direction of the reduction, suppose we have a family of k Rabin pairs $\mathcal{R}' = (\mathfrak{g}_i, \mathfrak{r}_i)_{1\le i\le k}$ (over some set of colours Γ') such that the automaton $\mathcal{A}'$ obtained by relabelling $\mathcal{A}$ with the acceptance condition $\mathsf{Rabin}(\mathcal{R}')$ recognises $\mathcal{L}(\mathcal{A})$. For all $v \in V$, as we assumed G to be connected, there is a finite path $\pi_v = v_{\mathsf{init}} e_1 v_1 e_2 v_2 \cdots e_\ell v$. We also define ρ_v as a finite pseudo-path $e_1 v e_2 \cdots v e_r v$ such that $\{e_1,\dots,e_r\} = \mathsf{adj}(v)$. The word $w_v = \pi_v \rho_v^\omega$ is in $\mathcal{L}(\mathcal{A})$, thus there exists $c_v \in \{1,\dots,k\}$ such that the run of w_v in $\mathcal{A}'$ satisfies $(\mathfrak{g}_{c_v}, \mathfrak{r}_{c_v})$. The set of colours it sees infinitely often is $\{v\} \cup \mathsf{adj}(v)$, thus $\mathfrak{r}_{c_v} \cap (\{v\} \cup \mathsf{adj}(v)) = \emptyset$ and $\mathfrak{g}_{c_v} \cap (\{v\} \cup \mathsf{adj}(v)) \neq \emptyset$. We thus define a function $\chi : V \to \{1,\dots,k\}$ as $v \mapsto c_v$.

It remains to prove that χ is a valid k-colouring of G. Suppose there exist two neighbours u, v such that $\chi(u) = \chi(v) = i$. Then the runs over w_u and w_v both satisfy $(\mathfrak{g}_i, \mathfrak{r}_i)$. As a result, the set $\{v\} \cup \mathsf{adj}(v) \cup \{u\} \cup \mathsf{adj}(u)$ also satisfies $(\mathfrak{g}_i, \mathfrak{r}_i)$. We define $\rho = e_1 u e_2 \cdots u e_r u$ and $\rho' = e_1' v e_2' \cdots v e_r' v$ with $e_1 = e_1' = \{u, v\}$. We can then observe that the word $\pi_u(\rho\rho')^\omega$ has an accepting run in $\mathcal{A}'$, as the colours it sees infinitely often are $\{v\} \cup \mathsf{adj}(v) \cup \{u\} \cup \mathsf{adj}(u)$. However, this word is not accepted by $\mathcal{A}_G$, a contradiction. As a result, χ is a valid k-colouring of G. ◀

This concludes our reduction, showing that the minimisation of Rabin pairs with respect to a given automaton is NP-hard.

5 Size of different representations of acceptance conditions

We start in Section 5.1 by analysing the size of the Zielonka tree and ACD in the worst case. Using Proposition 2.12, stating that minimal (history-)deterministic parity automata can be derived from the Zielonka tree, we can directly translate the lower bounds for the size of the Zielonka tree into lower bounds for (history-)deterministic parity automata. We recover in this way some results from Löding [30] and generalise them to history-deterministic automata. Then, we compare in Section 5.2 the size of different representations of Muller languages and study the translations between them, with special focus on the Zielonka tree and the Zielonka DAG, proving the claims from in Figure 3.

5.1 Worst case analysis of the Zielonka tree

We study the size of the Zielonka tree in the worst case. By Remark 2.16, the given bounds apply to the ACD, as the Zielonka tree can be seen as the ACD of a Muller automaton with just one state.

▶ **Proposition 5.1** (Size of the Zielonka tree: Worst case). *Let $\mathcal{F} \subseteq 2^{\Gamma}_{+}$ be a family of subsets, and let $m = |\Gamma|$. It holds:*

- $|\mathcal{Z}_{\mathcal{F}}| \leq 1 + m + m(m-1) + \cdots + m!$,
- $|\mathsf{Leaves}(\mathcal{Z}_{\mathcal{F}})| \leq m!$, *and*
- *the height of $\mathcal{Z}_{\mathcal{F}}$ is at most m.*

These bound are tight: for all $m \in \mathbb{N}$, there is a family $\mathcal{F}_m \subseteq 2^{\Gamma_m}_{+}$ over a set of m colours such that the previous relations are equalities.

Proof. We start by showing that the given bounds are tight. We suppose that m is even (the construction is symmetric if m is odd), and let $\Gamma_m = \{1, \dots, m\}$. Consider the family[6] $\mathsf{EvenLetters}_m \subseteq 2^{\Gamma_m}_{+}$ given by:

$$\mathsf{EvenLetters}_m = \{C \subseteq \Gamma_m \mid |C| \text{ is even}\}.$$

First, we remark that the last inequality follows from the fact that the subsets $\Gamma_m, \Gamma_m \setminus \{1\}, \dots, \Gamma_m \setminus \{1, \dots, m-1\}$ form a branch of the Zielonka tree. Let n be a node of the Zielonka tree of $\mathsf{EvenLetters}_m$, and let $X_n = \nu(n)$ be its label. Then n has a child for each subset of X_n of size $|X_n| - 1$. A simple induction gives that the level at depth k of the Zielonka tree has $m(m-1)\cdots(m-(k-1))$ nodes. This establishes the two first equalities of the statement.

We prove now the upper bounds. The last item follows from the fact that the label of a node is a set of size strictly smaller than the label of its parent. Let $\mathcal{F} \subseteq 2^{\Gamma}_{+}$, and $m = |\Gamma|$. We show by recurrence that the Zielonka tree $\mathcal{Z}_{\mathcal{F}}$ is not bigger than that of $\mathsf{EvenLetters}_m$. We remark that for all $X \subseteq \Gamma_m$, there is a node n_X in the Zielonka tree of $\mathsf{EvenLetters}_m$ labelled X, and that the subtree rooted at n_X is isomorphic to the Zielonka tree of $\mathsf{EvenLetters}_{|X|}$. Let n_0 be the root of $\mathcal{Z}_{\mathcal{F}}$, and let $n_1, \dots, n_k$ be its children. Then, we can find in the Zielonka tree of $\mathsf{EvenLetters}_m$ k incomparable nodes having as labels $\nu(n_1), \dots, \nu(n_k)$. By induction hypothesis, the subtree rooted at each of these nodes is not smaller than the subtrees rooted at $n_1, \dots, n_k$. This shows the two first items, ending the proof. ◀

We recover results analogous to those of Löding [30], and strengthen them as they apply to history-deterministic automata. These directly follow combining the previous proposition with Proposition 2.12.

▶ **Corollary 5.2.** *For every Muller language $L \subseteq \Gamma^{\omega}$ there exists a deterministic parity automaton recognising L of size at most $|\Gamma|!$. This bound is tight: for all n, a minimal history-deterministic parity automaton recognising the Muller language associated to $\mathsf{EvenLetters}_n$ has $n!$ states.*

Performing a slightly more careful analysis and using the characterisation of minimal history-deterministic Rabin automata by Casares, Colcombet and Lehtinen [10], we can obtain similar tight bounds for these automata. We refer to [7, Corollary II.93] for details.

---

[6] This family of subsets already appear in the worst-case study of parity automata recognising a Muller language in Mostowski's paper introducing the parity condition [36, p.161].

5.2 Comparing the sizes of various acceptance conditions

Colour-explicit vs Zielonka trees. First, we remark that a colour-explicit representation of a family $\mathcal{F}$ can be arbitrary larger than a representation of $\mathcal{Z}_{\mathcal{F}}$.

▶ **Proposition 5.3.** *For all $n \in \mathbb{N}$, there is a family of subsets $\mathcal{F}_n \subseteq 2_+^{\Gamma_n}$ over $\Gamma_n = \{1, \dots, n\}$ such that $|\mathcal{F}_n| = 2^n - 1$ and $|\mathcal{Z}_{\mathcal{F}}| = 1$.*

Proof. It suffices to take $\mathcal{F} = 2_+^{\Gamma_n}$. ◀

Even if the family $\mathcal{F}$ is represented explicitly as a list of subsets, we cannot compute its Zielonka tree in polynomial time, as $\mathcal{Z}_{\mathcal{F}}$ can be super-polynomially larger than $|\mathcal{F}|$.

▶ **Proposition 5.4.** *For all $n \in \mathbb{N}$, there is a family of subsets $\mathcal{F}_n \subseteq 2_+^{\Gamma_n}$ over $\Gamma_n = \{1, \dots, n\}$ such that:*

$$|\mathcal{Z}_{\mathcal{F}_n}| \geq |\mathcal{F}_n|^{\log(n)}.$$

More precisely, the family $\mathcal{F}_n$ satisfies $|\mathcal{Z}_{\mathcal{F}_n}| = n!$ and $|\mathcal{F}_n| = 2^n - 1$.

Proof. The family $\mathcal{F}_n = \mathsf{EvenLetters}_n$ from Proposition 5.1 satisfies $|\mathcal{Z}_{\mathcal{F}_n}| = n!$ and $|\mathcal{F}_n| = 2^n - 1$. The inequality $|\mathcal{Z}_{\mathcal{F}_n}| \geq |\mathcal{F}_n|^{\log(n)}$ is obtained by applying Stirling's formula. ◀

There is no apparent reason why the lower bound in the previous proposition should be optimal.

▶ Remark 5.5 (Question). Find tight bounds for the comparison of the size of a colour-explicit representation of a Muller language and the size of its Zielonka tree.

Colour-explicit vs Zielonka DAGs. Hunter and Dawar showed that we can compute the Zielonka DAG of a family $\mathcal{F}$ in polynomial time if $\mathcal{F}$ is given as a list of subsets [24, Theorem 3.17].

▶ **Proposition 5.6** ([24, Theorem 3.17])**.** *Given a family of subsets $\mathcal{F} \subseteq 2_+^{\Gamma}$, we can compute the Zielonka DAG of $\mathcal{F}$ in polynomial time in $|\mathcal{F}|+|\Gamma|$. In particular, $\mathcal{Z}$-$\mathsf{DAG}_{\mathcal{F}}$ has polynomial size in $|\mathcal{F}| + |\Gamma|$.*

However the reverse transformation cannot be done in polynomial time as Proposition 5.3 also applies to the Zielonka DAG.

Zielonka trees vs Zielonka DAGs. It is clear that, given a Zielonka tree $\mathcal{Z}_{\mathcal{F}}$, we can compute the corresponding Zielonka DAG $\mathcal{Z}$-$\mathsf{DAG}_{\mathcal{F}}$ in polynomial time. The converse is not possible. We note that this statement follows from complexity considerations: solving Muller games with the winning condition represented as a Zielonka DAG is PSPACE-complete [24], while solving those games with the condition represented as a Zielonka tree is equivalent to solving parity games [16], which can be done in quasi-polynomial time [5]. However, to the best of the author's knowledge, no explicit family witnessing an exponential gap between the two representations appears in the literature.

▶ **Proposition 5.7.** *For all $n \in \mathbb{N}$, there is a family of subsets $\mathcal{F}_n \subseteq 2_+^{\Gamma_n}$ over $\Gamma_n = \{1, \dots, n\}$ such that:*

- *the size of the Zielonka DAG of $\mathcal{F}_n$ is at most $2n$,*
- *the size of the Zielonka tree of $\mathcal{F}_n$ is at least $2^{\lfloor n/2 \rfloor}$.*

Proof. Consider the family defined as follows:

$$\mathsf{MinOddAndSucc}_n = \{C = \{c_1 < c_2 < \cdots < c_k\} \subseteq \Gamma_n \mid c_1 \text{ is odd and } c_2 = c_1 + 1\}.$$

Equivalently, we can describe this family as

$$\bigcup_{\substack{i=1,\\ i \text{ odd}}}^{n} X_i, \text{ where } X_i = \{C \subseteq \Gamma_n \mid i \in C \text{ and } i+1 \in C \text{ and } c > i \text{ for all } c \in C\}.$$

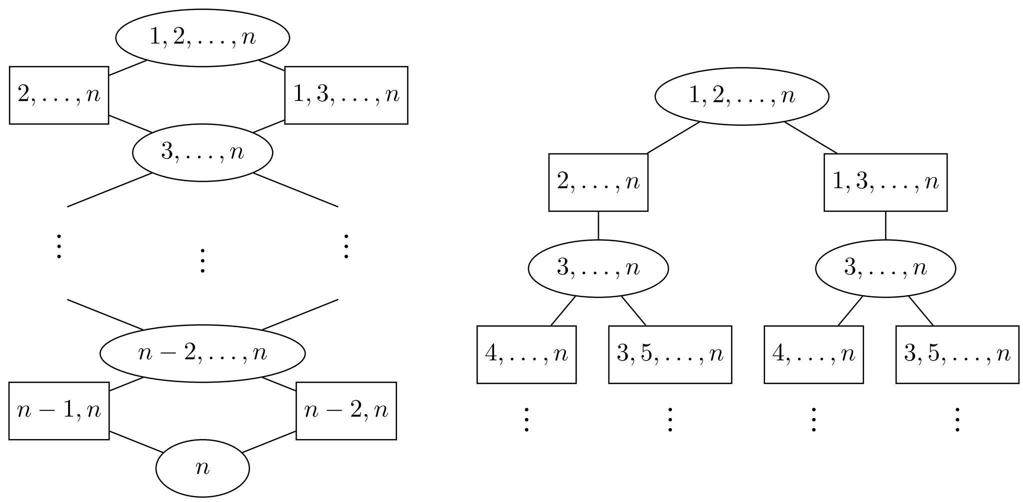


Figure 5 On the left, the Zielonka DAG of the condition $\mathsf{MinOddAndSucc}_n$ (for n odd), of size $\mathcal{O}(n)$. On the right, its Zielonka tree, of exponential size.

We show the Zielonka DAG and the Zielonka tree of $\mathsf{MinOddAndSucc}_n$ (for n odd) in Figure 5. We observe that the Zielonka DAG has height n; even levels consist in a single node, and odd levels have two nodes. Therefore, its size is $\lceil n/2 \rceil + n$. On the other hand, the Zielonka tree (with height also n), has $2^{\lfloor k/2 \rfloor}$ nodes at the level of depth k. ◀

Rabin vs Zielonka trees and Zielonka DAGs. If the Muller language associated to a family $\mathcal{F}$ is a Rabin language, then we can compute a family of Rabin pairs $\mathcal{R}$ such that $\mathsf{Rabin}(\mathcal{R}) = \mathsf{Muller}(\mathcal{F})$ in polynomial time. The converse is not possible, we cannot compute the Zielonka DAG in polynomial time, since it can be of exponential size in the number of Rabin pairs.

▶ **Proposition 5.8.** *Let $\mathcal{F} \subseteq 2^\Gamma_+$ be a family of subsets, and assume that $\mathsf{Muller}_\Gamma(\mathcal{F})$ is a Rabin language (that is, it admits a representation with Rabin pairs). Then, given the Zielonka DAG $\mathcal{Z}\text{-}\mathsf{DAG}_\mathcal{F}$ we can compute in polynomial time a family of Rabin pairs $\mathcal{R}$ over Γ such that $\mathsf{Rabin}_\Gamma(\mathcal{R}) = \mathsf{Muller}_\Gamma(\mathcal{F})$.*

Proof. Let $N = N_{\bigcirc} \sqcup N_{\square}$ be the nodes of the Zielonka DAG, partitioned into round and square nodes. By Proposition 6.2 from [9], all round nodes of $\mathcal{Z}\text{-}\mathsf{DAG}_\mathcal{F}$ have at most one child. We define a Rabin pair for each round node of $\mathcal{Z}\text{-}\mathsf{DAG}_\mathcal{F}$, $\mathcal{R} = \{(\mathfrak{g}_n, \mathfrak{r}_n)\}_{n \in N_{\bigcirc}}$, where $\mathfrak{g}_n$ and $\mathfrak{r}_n$ are defined as follows:

$$\begin{cases} \mathfrak{g}_n = \Gamma \setminus \nu(n), \\ \mathfrak{r}_n = \nu(n) \setminus \nu(n'), \text{ for } n' \text{ the only child of } n, \text{ if it exists.} \\ \mathfrak{r}_n = \nu(n) \text{ if } n \text{ has no children.} \end{cases}$$

That is, the pair $(\mathfrak{g}_n, \mathfrak{r}_n)$ accepts the sets of colours $A \subseteq \Gamma$ that contain some of the colours that disappear in the child of n and none of the colours appearing above n in the Zielonka DAG. We show that $\mathsf{Rabin}(\mathcal{R}) = \mathsf{Muller}(\mathcal{F})$. Let A be a set of colours. If $A \in \mathcal{F}$, let n be a maximal node (for $\preceq$) containing A. It is a round node and there is some colour $c \in A$ not appearing in the only child of n. Therefore, $c \in \mathfrak{g}_n$ and $A \cap \mathfrak{r}_n = \emptyset$. Conversely, if $A \notin \mathcal{F}$, then for every round node n with a child n', either $A \subseteq \nu(n')$ (and therefore $A \cap \mathfrak{g}_n = \emptyset$) or $A \nsubseteq \nu(n)$ (and in that case $A \cap \mathfrak{r}_n \neq \emptyset$). ◀

▶ **Proposition 5.9.** *For all $m \in \mathbb{N}$, there is a family $\mathcal{R}$ of m Rabin pairs over a set of colours Γ of size $2m$, such that $|\mathcal{Z}_{\mathcal{F}_\mathcal{R}}| \geq m!$ and $|\mathcal{Z}\text{-}\mathsf{DAG}_{\mathcal{F}_\mathcal{R}}| \geq 2^m$, where $\mathcal{F}_\mathcal{R} \subseteq 2^\Gamma_+$ is the (only) family such that $\mathsf{Muller}(\mathcal{F}_\mathcal{R}) = \mathsf{Rabin}(\mathcal{R})$.*

Proof. Let $\Gamma = \{g_1, r_1, g_2, r_2, \dots, g_m, r_m\}$ and define the Rabin pairs of $\mathcal{R}$ as $\mathfrak{g}_i = \{g_i\}$ and $\mathfrak{r}_i = \{r_i\}$. We depict the Zielonka tree of the corresponding family of subsets in Figure 6.

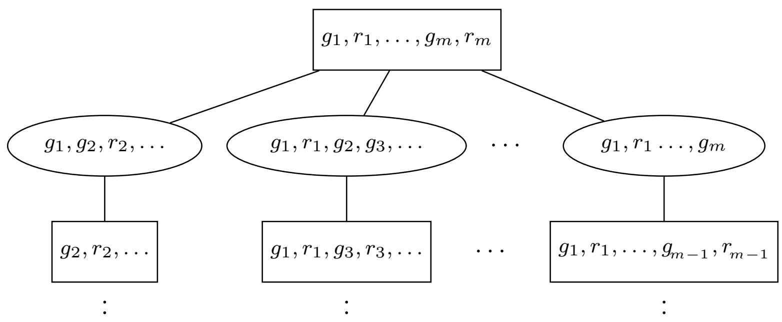


Figure 6 The Zielonka tree $\mathcal{Z}_{\mathcal{F}_\mathcal{R}}$ of the Rabin language from the proof of Proposition 5.9.

The Zielonka tree $\mathcal{Z}_{\mathcal{F}_\mathcal{R}}$ satisfies that the levels at depth k and $k+1$ have $m(m-1)\dots k$ nodes, which shows that $|\mathsf{Leaves}(\mathcal{Z}_{\mathcal{F}_\mathcal{R}})| = m!$. For the bound on the size of the Zielonka DAG, we observe that for each subset $X \subseteq \{1, \dots, m\}$ there is at least one subset appearing as the label of some nodes of the Zielonka tree, namely, $\{g_i, r_i \mid i \in X\}$. ◀

6 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 3.1 and 3.3). 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 [11].

Furthermore, this result has several implications for our simplification purpose:

- We can decide the typeness of Muller automata in polynomial time (Corollary 3.4).
- We can compute the parity index of a language recognised by a deterministic Muller automaton in polynomial time (Corollary 3.5).

In addition, we showed that we can minimise in polynomial 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.

In sum, our results help to clarify the potential of the alternating cycle decomposition and complete the picture of our understanding about the possibility of simplifying the acceptance conditions of ω-automata.

References

1 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.

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

3 Endre Boros and Ondrej Cepek. On the complexity of Horn minimization. *Rutgers University. Rutgers Center for Operations Research [RUTCOR]*, 1994.

4 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.

5 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.

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

7 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.

8 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.

9 Antonio Casares, Thomas Colcombet, Nathanaël Fijalkow, and Karoliina Lehtinen. From Muller to parity and Rabin automata: Optimal transformations preserving (history-)determinism. *CoRR*, abs/2305.04323, 2023. doi:10.48550/arXiv.2305.04323.

10 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.

11 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.

12 Tom Chang. Horn formula minimization, 2006. Available at https://scholarworks.rit.edu/cgi/viewcontent.cgi?article=7895&context=theses.

13 Edmund M. Clarke, I. A. Draghicescu, and Robert P. Kurshan. A unified approch for showing language inclusion and equivalence between various types of omega-automata. *Inf. Process. Lett.*, 46(6):301–308, 1993. doi:10.1016/0020-0190(93)90069-L.

14 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.

15 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.

16 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.

17 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.

18 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.

19 M.R. Garey, D.S. Johnson, and L. Stockmeyer. Some simplified NP-complete graph problems. *Theoretical Computer Science*, 1(3):237–267, 1976. URL: https://www.sciencedirect.com/science/article/pii/0304397576900591, doi:https://doi.org/10.1016/0304-3975(76)90059-1.

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 Florian Horn. Explicit Muller games are PTIME. In *FSTTCS*, pages 235–243, 2008. doi:10.4230/LIPIcs.FSTTCS.2008.1756.

22 Christopher Hugenroth. Zielonka DAG acceptance, regular languages over infinite words. In *DLT*, 2023.

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

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

25 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.

26 Richard M. Karp. *Reducibility among Combinatorial Problems*, pages 85–103. The IBM Research Symposia Series. Springer US, 1972. doi:10.1007/978-1-4684-2001-2_9.

27 Jan Kretínský, 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 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.

29 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.

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

31 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.

32 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.

33 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 [8], 2021.

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

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

36 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.

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

38 Damian Niwiński and Igor Walukiewicz. Relating hierarchies of word and tree automata. In *STACS*, pages 320–331, 1998. doi:10.1007/BFb0028571.

39 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.

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

41 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.

42 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.

43 Robert Tarjan. Depth first search and linear graph algorithms. *Siam Journal On Computing*, 1(2), 1972. doi:10.1137/0201010.

44 Cong Tian and Zhenhua Duan. Büchi determinization made tighter. *CoRR*, abs/1404.1436, 2014. URL: http://arxiv.org/abs/1404.1436.

45 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.

A Generalised 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 [14].

In this appendix, we study 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 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.

We will prove that we can minimise the number of clauses in a GH formula in polynomial-time, using our algorithm for minimising the number of pairs in a Rabin condition as a black box.

This result contrasts nicely with the NP-completeness of minimising the number of clauses in a Horn formula [3] (see also [12]). 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 [12] to GH formulas.

Our technique for clause minimisation may thus be of interest for the study of Horn formulas.

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 A.4). 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.

▶ **Definition A.1.** *A Horn clause is a disjunction of literals with at most one* positive *literal, that is, a literal with no negation. Equivalently, it is a boolean formula of the form either $(x_1 \wedge \dots \wedge x_n) \implies y$ or $(x_1 \wedge \dots \wedge x_n) \implies \bot$. A Horn formula is a conjunction of Horn clauses.*

A generalised Horn clause (or GH clause) is a boolean formula of the form either $(x_1 \wedge \dots \wedge x_n) \implies (y_1 \wedge \dots \wedge y_m)$ or $(x_1 \wedge \dots \wedge x_n) \implies \bot$ (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.

We will now use our PTIME algorithm for minimising the number of pairs in a Rabin condition to minimise the number of clauses in a GH formula (Proposition A.3). We start by applying it to minimise the number of clauses of simple GH formulas.

In all that follows we will not distinguish valuations $\nu : \mathrm{Var} \to \{\top, \bot\}$ from the corresponding subsets of variables $\{v \in \mathrm{Var} \mid \nu(v) = \top\}$.

▶ **Lemma A.2.** *There is a polynomial-time algorithm that minimises the number of clauses of a simple GH formula.*

Proof. It suffices to observe that there is a correspondence between simple GH formulas and Streett conditions. Define the function α that turns a GH clause $(x_1 \wedge \dots \wedge x_n) \implies (y_1 \wedge \dots \wedge y_m)$ into the Rabin pair $(\{y_1, \dots, y_m\}, \{x_1, \dots, x_n\})$. We extend it into a function turning simple GH formulas into families of Rabin pairs by defining $\alpha(\bigwedge_{i=1}^{k} \mathrm{GH}_i) = (\alpha(\mathrm{GH}_i))_{i=1}^{k}$, with its associated Streett language. 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.

Finally, note that for all simple GH formula φ the set of sets accepted by the Streett condition $\alpha(\varphi)$ is $\{\nu^{-1}(\bot) \mid \nu \text{ satisfies } \varphi\}$. As a result, two simple GH formula are equivalent if and only if their images by α define the same Streett language.

In conclusion, in order to minimise the number of clauses of a simple GH formula, one can simply apply α to it, minimise the number of pairs in the resulting Streett condition, and then apply α^{-1}. ◀

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 previous proof. We circumvent this problem by replacing them with some non-negative clauses and proving that minimising the initial Horn formula comes down to minimising the resulting simple one.

▶ **Proposition A.3.** *There is a polynomial-time algorithm to minimise the number of clauses of a GH formula.*

Proof. Let φ be a GH formula, V the set of variables appearing in it. If φ does not contain any negative clause, then it is satisfied by the valuation mapping every variable to $\top$ and thus can only be equivalent to simple GH formulas. We can thus apply Lemma A.2 directly.

Let ψ be a simple GH formula and $N_1, \dots, N_k$ negative Horn clauses, with $k > 0$, such that $\varphi = \psi \wedge \neg N_1 \wedge \dots \wedge \neg N_k$. We add a fresh variable $x_\bot$ that will play the role of $\bot$. For all $i \in [1, k]$, let $x_1^i, \dots, x_{p(i)}^i$ be such that $N_i = (x_1^i \wedge \dots \wedge x_{p(i)}^i \implies \bot)$ and let $C_i = (x_1^i \wedge \dots \wedge x_{p(i)}^i \implies x_\bot)$. Define $\tilde{\varphi} = \psi \wedge C_1 \wedge \dots \wedge C_k \wedge (x_\bot \Rightarrow \bigwedge_{y \in V} y)$. Note that the valuations satisfying $\tilde{\varphi}$ are exactly the ones mapping $x_\bot$ to $\bot$ and whose projection on the other variables satisfies φ, plus the one mapping every variable to $\top$.

As $\tilde{\varphi}$ is simple, we can apply Lemma A.2 to obtain an equivalent simple GH formula $\tilde{\varphi}_{\min}$ with a minimal number of clauses. We define $\varphi_{\min}$ as this formula where every clause with $x_\bot$ on the left side has been removed and every clause of the form $(x_1 \wedge \dots \wedge x_n) \implies (y_1 \wedge \dots \wedge y_m)$ where one of the y_i is $x_\bot$ has been replaced by $(x_1 \wedge \dots \wedge x_n \implies \bot)$.

As $\tilde{\varphi}$ is not satisfied by the valuation mapping $x_\bot$ to $\top$ and all other variables to $\bot$, at least one clause in $\tilde{\varphi}_{\min}$ has an $x_\bot$ on the left, hence $\varphi_{\min}$ has less clauses than $\tilde{\varphi}_{\min}$.

We have to argue that φ and $\varphi_{\min}$ are equivalent, and that $\varphi_{\min}$ is minimal with respect to the number of clauses. First let us show that φ and $\varphi_{\min}$ are equivalent. Let ν be a valuation, we write $\nu_\bot$ for the valuation mapping $x_\bot$ to $\bot$ and matching ν on V. We have

$$\nu \text{ satisfies } \varphi_{\min} \iff \nu_\bot \text{ satisfies } \tilde{\varphi}_{\min} \iff \nu_\bot \text{ satisfies } \tilde{\varphi} \iff \nu \text{ satisfies } \varphi.$$

Then let us prove that $\varphi_{\min}$ is minimal with respect to the number of clauses. Assume by contradiction that we have a GH formula φ' equivalent to φ and with less clauses than $\varphi_{\min}$. Then we can replace every negative clause $(\neg x_1 \vee \dots \vee \neg x_n)$ in φ' by a clause $(x_1 \wedge \dots \wedge x_n) \implies x_\bot$ and add a clause $(x_\bot \implies \bigwedge_{y \in V} y)$ to get a simple GH formula φ'' equivalent to $\tilde{\varphi}$ and with less clauses than $\tilde{\varphi}_{\min}$. This contradicts the minimality of $\tilde{\varphi}_{\min}$.

Hence $\varphi_{\min}$ has a minimal number of clauses. ◀

Given a GH formula φ using variables in Γ, its *GH language* is

$$\mathsf{GH}_\Gamma(\varphi) = \{w \in \Gamma^\omega \mid \mathsf{Inf}(w) \models \varphi\}.$$

▶ **Proposition A.4.** *Checking emptiness of an automaton with an acceptance condition represented by a GH formula is* NP*-complete.*

Proof. The NP upper bound follows from the one on Emerson-Lei conditions.

For the hardness, we reduce from the Hamiltonian cycle problem. Let $G = (V, E)$ be a directed graph. For all edge $e \in E$ we write $\mathrm{src}(e)$ for its first vertex and $\mathrm{tgt}(e)$ for the second one. We define the automaton $\mathcal{A} = (Q, q_{\mathsf{init}}, \Sigma, \Delta, \Gamma, \mathsf{col}, W)$ as follows:

- $Q = \{v^-, v^+ \mid v \in V\}$, and we pick an arbitrary $v \in V$ and set $q_{\mathsf{init}} = v^-$.
- $\Sigma = \Gamma = \{l_v \mid v \in V\} \cup \{l_e \mid e \in E\} \cup \{l_\bot\}$, every transition is coloured with the letter it reads.
- $\Delta = \{(v^-, l_v, v^+) \mid v \in V\} \cup \{(\mathrm{src}(e)^+, l_e, \mathrm{tgt}(e)^-) \mid e \in E\}$.
- $W = \mathsf{GH}_\Gamma(\varphi)$ with

$$\varphi = \Big[\bigwedge_{\substack{e \neq e' \in E \\ \mathrm{src}(e) = \mathrm{src}(e')}} (l_e \wedge l_{e'} \implies l_\bot) \Big] \wedge \Big[\bigwedge_{v \in V} (l_v \implies \bigwedge_{v' \in V} l_{v'}) \Big].$$

A run of $\mathcal{A}$ is a sequence $v_0^- \xrightarrow{v_0} v_0^+ \xrightarrow{(v_0,v_1)} v_1^- \xrightarrow{v_1} v_1^+ \xrightarrow{(v_1,v_2)} \cdots$. It is accepted if and only if all vertices are visited infinitely often and the run ultimately always selects the same edge from every vertex. The existence of such a run is equivalent to the existence of a Hamiltonian cycle. ◀