

Optimal Transformations of Muller Conditions

Antonio Casares 

LaBRI, Université de Bordeaux, France
antonio.casares-santos@labri.fr

Thomas Colcombet 

CNRS, IRIF, Université de Paris, France
thomas.colcombet@irif.fr

Nathanaël Fijalkow 

CNRS, LaBRI, Université de Bordeaux, France
The Alan Turing Institute of Data Science, London, United Kingdom
nathanael.fijalkow@labri.fr

Abstract

In this paper we are interested in automata over infinite words and infinite duration games, that we view as general transition systems. We study transformations of systems using a Muller condition into ones using a parity condition, extending Zielonka’s construction. We introduce the alternating cycle decomposition transformation, and we prove a strong optimality result: for any given deterministic Muller automaton, the obtained parity automaton is minimal both in size and number of priorities among those automata admitting a morphism into the original Muller automaton.

We give two applications. The first is an improvement in the process of determinisation of Büchi automata into parity automata by Piterman and Schewe. The second is to present alternative proofs unifying several results about the possibility of relabelling deterministic automata with different conditions.

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

Keywords and phrases Automata over infinite words, Omega regular languages, Determinisation of automata

1 Introduction

Automata over infinite words were first introduced in the 60s by Büchi [1], in his proof of the decidability of the monadic second order theory of the natural numbers with successor $(\mathbb{N}, 0, succ)$. Contrary to automata over finite words, there is not a unique natural definition for acceptance in the infinite setting. The condition used by Büchi (called [Büchi condition](#)), accepts those runs that visit infinitely often a final state. The [Muller condition](#), introduced in [18], specifies a family of subsets of states and accepts a run if the states visited infinitely often form a set in that family. Other acceptance conditions have been defined, and in particular the [parity condition](#), introduced by Mostowski in [17], is of notable importance. It has the same expressive power as the Muller condition and it is specially well-behaved and adequate for algorithmic manipulation. For example, it allows easy complementation of automata, it admits memoryless strategies for games, and parity game solvers have been proved very efficient. The [parity condition](#) assigns to each edge of an automaton a natural number, called a priority, and a run is accepting if the smallest priority visited infinitely often is even. Therefore, another important parameter dealing with parity conditions is the number of different priorities that it uses. An automaton using an acceptance condition of type \mathcal{C} is called a \mathcal{C} automaton.

As it is to be expected, different acceptance conditions have different expressive power. It is not difficult to see that non-deterministic Büchi automata have strictly more expressive power than deterministic ones. However, McNaughton theorem [16] states that Büchi automata

can be transformed into deterministic Muller automata accepting the same language. Non-deterministic Büchi and Muller automata are also equivalent to regular expressions and MSO logic over infinite words. Deterministic parity automata stand out as the simplest type of deterministic automata being equivalent to all these models [17].

In this work we study methods which transform automata and games that use Muller acceptance conditions into others using parity acceptance conditions. We present the constructions using the formalism of [transition systems](#) in order to obtain the most general results. These constructions can be immediately applied to transform deterministic or non-deterministic, as well as games. We work with *transition-labelled* systems (instead of state-labelled) for technical convenience.

The standard way to transform a Muller transition system into a parity transition system is to build a deterministic parity automaton that [recognizes](#) the Muller condition, and then take the [product](#) of the Muller transition system and the automaton, which is a transition system using a parity condition. We can find the first example of one such automaton implicitly in the work of Gurevich and Harrington [9], called a *later appearance record (LAR)*. The ideas of Gurevich and Harrington have recently been refined in order to find smaller automata [15, 13]. On the other hand, in his work on the memory requirements of Muller games [27], Zielonka presents the notion of the *split tree* associated to a [Muller condition](#) (later called [Zielonka tree](#) [8, 11]). This construction yields another [parity automaton](#) (that we call [Zielonka tree automaton](#)) recognizing a Muller condition, and it is made explicit in section 3. We prove in section 3 that the Zielonka tree automaton is a minimal parity automaton recognizing a Muller condition and it uses a parity condition with the minimal number of priorities.

However, the [product](#) of a Muller transition system \mathcal{T} and the [Zielonka tree automaton](#) does not result in general in the best parity transition system simulating \mathcal{T} . In order to optimise this transformation, we present in section 4 a data structure, the [alternating cycle decomposition \(ACD\)](#) of \mathcal{T} , that incorporates the information of how the transition system \mathcal{T} makes use locally of the Muller condition. The alternating cycle decomposition is obtained applying the Zielonka tree construction while taking into account the structure of \mathcal{T} by considering the loops that are alternatively accepting and rejecting. The idea of considering the alternating chains of loops of an automaton is already present in the work of Wagner [26]. In section 4 we prove that the transformation given by the [alternating cycle decomposition](#) is optimal and it uses a [parity condition](#) with the optimal number of priorities.

In this report we are concerned with *state complexity*, this is, the efficiency of a construction is measured based on the number of states of the resulting transition system. However, we emphasize that the word *optimal* might be used in a stronger sense than usual. When we say that a transformation is optimal we do not only mean that in the worst case it produces a minimal transition system, but that *in all cases* we obtain a minimal transition system with the desired properties. For instance, it is sometimes stated that the *LAR* automaton of [9] is optimal (see [15]), however, it only is in the worst case, as in most other cases the [Zielonka tree automaton](#) has strictly smaller size.

At the end of this report we study the applicability of the proposed transformation to one of the main concerns in many applications of automata over infinite words (as for instance the synthesis for LTL formulas): the determinisation of [Büchi automata](#). The first efficient determinisation procedure was proposed by Safra [21], and since then many other constructions have been proposed. In [20], Piterman proposed a modification on Safra's construction that improves the complexity and directly produces a [parity automaton](#). In [22], Schewe revisits Piterman's construction differentiating two steps: a first one producing

a deterministic [Rabin](#) automaton, and a second one producing a [parity](#) automaton.

In [6], Colcombet and Zdanowski found a tight worst-case lower bound for the first step, and Schewe and Varghese found a tight (up to a constant) worst-case lower bound for the second step in [25, 23]. The [alternating cycle decomposition](#) presented in this report provides a new procedure to transform the Rabin automaton to a parity one that improves the construction proposed in [22] (theorem 5.6).

Contributions.

In this report, we establish four results:

Optimal parity automata for Muller conditions. We present how to use the [Zielonka tree](#) of a [Muller condition](#) in order to build a parity automaton [recognizing](#) this condition. We prove that this automaton is optimal for every [Muller condition](#), both in terms of number of priorities used (proposition 3.9) and of size (theorem 3.13). This result is new here, but this construction can be considered as already known by the community.

Optimal transformation of Muller into parity transition systems. We provide a construction translating Muller transition systems into parity transition systems, that can be seen as a generalisation of the above one. We introduce the [alternating cycle decompositions](#) which are generalisations of [Zielonka trees](#) to transition systems, and derive from them our [alternating cycle decomposition transformation](#) (ACD-transformation).

The optimality part of this proof uses the concept of [locally bijective morphism](#): when the ACD-transformation is applied to a Muller transition system \mathcal{T} , it outputs a parity transition system \mathcal{T}' such that there exists a [locally bijective morphism](#) from \mathcal{T}' to \mathcal{T} . The optimality result states that \mathcal{T}' has the minimum number of states such that this property holds (theorem 4.26), and that its parity condition uses an optimal number of priorities (proposition 4.24).

Improvement on Piterman-Schewe's determinisation of Büchi automata. Piterman and Schewe have described an efficient translation from non-deterministic Büchi automata to deterministic parity automata [20, 22] (both are variations around the famous construction of Safra [21]). Schewe describes this construction as first building a Rabin automaton \mathcal{R}_B , followed by an ad-hoc transformation of this automaton for producing a parity automaton \mathcal{P}_B . This second step induces in fact a [locally bijective morphism](#). This implies that we obtain a smaller parity automaton by applying the [ACD-transformation](#) to \mathcal{R}_B . We also provide an example where this automaton is indeed strictly smaller and uses less priorities (example 5.7).

New proofs for results of automata relabelling. Finally, we show how the [alternating cycle decomposition](#) directly implies two existing results: if a deterministic automaton can be labelled by a [Rabin condition](#) and by a [Streett condition](#) while accepting the same language, then it can be labelled by a [parity condition](#) while accepting the same language [3], and if a deterministic automaton can be labelled by a Büchi condition and by a co-Büchi condition while accepting the same language, then it can be labelled by a [weak condition](#) while accepting the same language.

Organisation of this report.

In section 2 we present the definitions and notations that we will use throughout the report.

In section 3 we define the [Zielonka tree](#) and the [Zielonka tree automaton](#) and we prove the optimality of the latter. In section 3.3 we present some examples of [Zielonka trees](#) for special acceptance conditions, that will be useful in section 5.2. Most constructions and

proofs of this section can be regarded as special cases of those from section 4. However, we find instructive to include them separately since this is the opportunity to describe all the core ideas that will appear in section 4 in a simpler setting.

We begin section 4 by defining [locally bijective morphisms](#). In section 4.2 we present the main contribution of this work: the [alternating cycle decomposition](#), and we prove its optimality in section 4.4.

Section 5 is divided in two very different parts. In section 5.1 we show how the [ACD-transformation](#) could provide a smaller deterministic parity automaton than the constructions of [20] and [22]. In section 5.2 we analyse the information given by the [alternating cycle decomposition](#) and we provide two original proofs concerning the possibility of labelling automata with different acceptance conditions.

We have included detailed examples all throughout the report. We hope that these will help the reader to better understand the sometimes intricate formalism.

2 Notations and definitions

In this chapter we introduce standard notions that will be used throughout the report. We begin with some basic notations in section 2.1.

2.1 Basic notations

We let $\mathcal{P}(A)$ denote the power set of a set A and $|A|$ its cardinality. The symbol ω denotes the ordered set of non-negative integers. For $i, j \in \omega$, $i < j$, $[i, j]$ 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 length of a word u is $|u|$. An [\$\omega\$ -word](#) (or simply an [infinite word](#)) is a word of length ω . The sets of finite and infinite words over Σ will be written Σ^* and Σ^ω respectively. We let $\Sigma^\infty = \Sigma^* \cup \Sigma^\omega$. For a word $u \in \Sigma^\infty$ we write $u(i)$ or u_i to represent the i -th letter of u . We let ε denote the [empty word](#). For $u \in \Sigma^*$ and $v \in \Sigma^\infty$, the concatenation of these words is written $u \cdot v$, or simply uv . If $u = v \cdot w$ for $v \in \Sigma^*$, $u, w \in \Sigma^\infty$, we say that v is a [prefix](#) of u and we write $v \sqsubseteq u$ (it induces a partial order on Σ^*).

For a finite [word](#) $u \in \Sigma^*$ we write $\text{First}(u) = u(0)$ and $\text{Last}(u) = u(|u| - 1)$. For a word $u \in \Sigma^\infty$, we let $\text{Inf}(u) = \{a \in \Sigma : u(i) = a \text{ for infinite } i \in \omega\}$ and $\text{App}(u) = \{a \in \Sigma : \exists i \in \omega \text{ such that } u(i) = a\}$.

Given a map $\alpha : A \rightarrow B$, we will extend α to words component-wise. I.e., $\alpha : A^\infty \rightarrow B^\infty$ will be defined as $\alpha(a_0 a_1 a_2 \dots) = \alpha(a_0) \alpha(a_1) \alpha(a_2) \dots$.

A [directed graph](#) is a tuple $(V, E, \text{Source}, \text{Target})$ where V is a set of vertices, E a set of edges and $\text{Source}, \text{Target} : E \rightarrow V$ are maps indicating the source and target for each edge. A [path](#) from $v_1 \in V$ to $v_2 \in V$ is a word $\varrho \in E^*$ such that $\text{First}(\varrho) = v_1$, $\text{Last}(\varrho) = v_2$ and $\text{Source}(\varrho_i + 1) = \text{Target}(\varrho_i)$ for $i < |\varrho|$. A graph is [strongly connected](#) if there is a path connecting each pair of vertices of it. A [subgraph](#) of $(V, E, \text{Source}, \text{Target})$ is a graph $(V', E', \text{Source}', \text{Target}')$ such that $V' \subseteq V$, $E' \subseteq E$ and Source' and Target' are the restriction to E' of Source and Target , respectively. A [strongly connected component](#) is a maximal [strongly connected](#) subgraph.

2.2 Automata over infinite words

A [non-deterministic automaton](#) (which we will simply call an *automaton*) is a tuple $\mathcal{A} = (Q, \Sigma, I_0, \Gamma, \delta, \text{Acc})$ where

- Q is a set of states.

- Σ is an input alphabet.
- $I_0 \subseteq Q$ is a non-empty set of initial states.
- Γ is an output alphabet.
- $\delta : Q \times \Sigma \rightarrow \mathcal{P}(Q \times \Gamma)$ is a transition function.
- $Acc \subseteq \Gamma^\omega$ is an acceptance condition.

If for every $q \in Q$, $a \in \Sigma$, $\delta(q, a) \neq \emptyset$ we say that the automaton is **Σ -complete**. We can always suppose that an automaton is Σ -complete by adding a “sink node” s to Q that receives the not previously defined transitions.

If I_0 is a singleton and for every $q \in Q$, $a \in \Sigma$, $\delta(q, a)$ is a singleton, we say that \mathcal{A} is a **deterministic automaton** (in particular a deterministic automaton is **Σ -complete** for us). In this case we will split the transition function into $\delta : Q \times \Sigma \rightarrow Q$ and $\gamma : Q \times \Sigma \rightarrow \Gamma$. In some cases we will omit the output alphabet Γ . If so, we will implicitly take as the output alphabet the whole set of transitions, $\Gamma = \{(q, a, \delta(q, a)) : q \in Q, a \in \Sigma\}$. (See figure 1 for examples).

We extend the definition of δ to finite words $\delta : Q \times \Sigma^* \rightarrow Q$ recursively:

- $\delta(q, \varepsilon) = q$, for $q \in Q$
- $\delta(q, wa) = \delta(\delta(q, w), a)$, for $w \in \Sigma^*$ and $a \in \Sigma$

Given an **automaton** \mathcal{A} and a word $u \in \Sigma^\omega$, a **run over u** in \mathcal{A} is a sequence

$$\varrho = (q_0, u_0, b_0, q_1)(q_1, u_1, b_1, q_2) \dots \quad q_i \in Q, u_i \in \Sigma, b_i \in \Gamma \text{ for every } i \in \omega$$

such that $q_0 \in I_0$ and $(q_{i+1}, b_i) = \delta(q_i, u_i)$ for all $i \in \omega$. The **output** of the run ϱ is the word $Output_{\mathcal{A}}(\varrho) = b_0 b_1 b_2 \dots \in \Gamma^\omega$. The word u is **accepted** by \mathcal{A} if it exists a run over u , ϱ , such that $Output_{\mathcal{A}}(\varrho) \in Acc$.

The **language accepted** (or recognized) by an automaton \mathcal{A} is the set

$$\mathcal{L}(\mathcal{A}) := \{u \in \Sigma^\omega : u \text{ is accepted by } \mathcal{A}\}$$

We remark that if \mathcal{A} is **deterministic** then there is a single **run over u** for each $u \in \Sigma^\omega$. We let $\mathcal{A}(u)$ denote the **output** of this run.

► **Remark.** We have defined transition-labelled automata (the acceptance condition is defined over transitions instead of over states). Nevertheless, transition-labelled automata are easily transformed into state-labelled automata.

2.3 Transition systems

A **transition system graph** $\mathcal{T}_G = (V, E, Source, Target, I_0)$ is a **directed graph** with a set of initial vertices, where

- V is a set of vertices (also called states) and E is a set of edges (also called transitions).
- $Source : E \rightarrow V$ associates an edge with its source vertex.
- $Target : E \rightarrow V$ associates an edge with its target vertex.
- $I_0 \subseteq V$ is a non-empty set of initial vertices.

We will suppose in this work that every vertex of a transition system graph has at least one outgoing edge.

A **transition system** \mathcal{T} is obtained from a transition system graph \mathcal{T}_G adding:

- A colouring of the edges $\gamma : E \rightarrow \Gamma$.
- An **acceptance condition** $Acc \subseteq \Gamma^\omega$.

We will usually take $\Gamma = E$ and γ the identity function. In that case we will omit the set of colours.

A *run* from $q \in V$ on a *transition system graph* \mathcal{T} is a sequence of edges $\varrho = e_0 e_1 \dots \in E^\infty$ such that $Source(e_0) = q$ and $Target(e_i) = Source(e_{i+1})$ for all $i < |\varrho|$.

For $A \subseteq V$ we let $\mathcal{Run}_{\mathcal{T}, A}$ denote the set of runs on \mathcal{T} starting from some $q \in A$ (we omit brackets if $A = \{q\}$), and $\mathcal{Run}_{\mathcal{T}} = \mathcal{Run}_{\mathcal{T}, I_0}$ the set of runs starting from some initial vertex.

A run $\varrho \in \mathcal{Run}_{\mathcal{T}}$ is *accepting* if $\gamma(\varrho) \in Acc$. A run $\varrho \in \mathcal{Run}_{\mathcal{T}}$ is *rejecting* if it is not accepting.

We say that a vertex $v \in V$ is *accessible* (or *reachable* if there is a finite run $\varrho \in \mathcal{Run}_{\mathcal{T}}$ (starting from an initial vertex) such that $Last(\varrho) = v$.

Given a transition system \mathcal{T} we let $|\mathcal{T}|$ denote $|V|$ for V its set of vertices. For a subset of vertices $A \subseteq V$ we write:

- $In(A) = \{e \in E : Target(e) \in A\}$
- $Out(A) = \{e \in E : Source(e) \in A\}$

We might want to add more information to a transition system. A *labelled transition system* is a *transition system* \mathcal{T} with labelling functions $l_V : V \rightarrow L_V$, $l_E : E \rightarrow L_E$ into sets of labels for vertices and edges respectively.

► **Example 2.1 (Automata as transition systems).** An *automaton* $\mathcal{A} = (Q, \Sigma, I_0, \delta, Acc)$ can be seen as a *labelled transition system* $\mathcal{T} = (V, E, Source, Target, I_0, Acc, l_E)$, taking $V = Q$, $E = \{(q, a, q') : q \in Q, a \in \Sigma, \delta(q, a) = (q', b)\}$, $Source$ and $Target$ the projections into the first and last component respectively and adding labels indicating the input letters:

$$l_E : E \rightarrow \Sigma ; l_E(q, a, b, q') = a$$

The automaton \mathcal{A} is *deterministic* if and only if from every vertex $v \in V$ and $a \in \Sigma$ there exists a unique edge $e \in Out(v)$ such that $l_E(e) = a$ and I_0 is a singleton.

Depending on the context, we will use one formalism or the other to work with automata, both of them being completely equivalent.

► **Definition 2.2 (Games).** A *game* $\mathcal{G} = (V, E, Source, Target, v_0, Acc, l_V)$ is a *transition system* with a single initial vertex v_0 and vertices labelled by a function $l_V : V \rightarrow \{Eve, Adam\}$ that induces a partition of V into vertices controlled by a player named Eve and another named Adam.

During a *play*, players move a token from one vertex to another, starting from the initial vertex v_0 . The player who owns the vertex v where the token is placed chooses an edge in $Out(v)$ and the token travels through this edge to its target. In this way, they produce an infinite *run* ϱ over \mathcal{G} (that we also call a play). We say that Eve wins the play if it belongs to the acceptance condition Acc (and Adam wins in the contrary).

A *strategy* for a player $P \in \{Eve, Adam\}$ is a function $S_P : \mathcal{Run}_{\mathcal{G}} \cap E^* \rightarrow E$ that tells the player which move to choose after a finite play. We say that a *play* $\varrho \in \mathcal{Run}_{\mathcal{G}}$ is consistent with the strategy S_P for player P if after each finite subplay $\varrho' \sqsubseteq \varrho$ ending in a vertex controlled by P , the next edge in ϱ is $S_P(\varrho')$. We say that Eve *wins* the game \mathcal{G} if there is a strategy S_{Eve} such that all plays consistent with S_{Eve} for Eve are accepted. Dually, Adam wins \mathcal{G} if there is a strategy S_{Adam} such that no play consistent with S_{Adam} for Adam is accepted.

Given a game \mathcal{G} , the *winning region* of \mathcal{G} for player $P \in \{Eve, Adam\}$, written $\mathcal{W}_P(\mathcal{G})$, is the set of vertices $v \in V$ such that P wins the game \mathcal{G}' obtained by setting the initial vertex to v in \mathcal{G} .

Product of a transition system and an automaton

Let $\mathcal{T} = (V, E, \text{Source}, \text{Target}, I_0, \sigma : E \rightarrow \Sigma)$ be a [transition system graph](#) with transitions coloured by colours in a set Σ , and let $\mathcal{A} = (Q, \Sigma, q_0, \Gamma, \delta, \gamma, \text{Acc})$ be a [deterministic automaton](#) over the alphabet Σ . We define the [product](#) of \mathcal{T}_G and \mathcal{A} as the transition system $\mathcal{T}_G \times \mathcal{A} = (V \times Q, E', \text{Source}', \text{Target}', I_0 \times \{q_0\}, \gamma', \text{Acc})$, where:

- The set of vertices is the cartesian product $V \times Q$.
- The set of edges is $E' = E \times Q$.
- $\text{Source}'(e, q) = (\text{Source}(e), q)$.
- $\text{Target}'(e, q) = (\text{Target}(e), \delta(q, \sigma(e)))$.
- The initial set is $I_0 \times \{q_0\}$.
- There is a natural [acceptance condition](#) given by the colouring $\gamma' : E \times Q \rightarrow \Gamma$, $\gamma'(e, q) = \gamma(q, \sigma(e))$ and the set $\text{Acc} \subseteq \Gamma^\omega$.

Intuitively, a computation in $\mathcal{T} \times \mathcal{A}$ happens as follows: we start from a vertex $v_0 \in I_0$ in \mathcal{T} and from q_0 in Q . Whenever a transition e between v_1 and v_2 takes places in \mathcal{T} , it produces the colour $c(e) \in \Sigma$. Then, the automaton \mathcal{A} makes the transition corresponding to $\sigma(e)$, producing an output in Γ . In this way, a word in Γ^ω is produced and we can use the [acceptance condition](#) $\text{Acc} \subseteq \Gamma^\omega$ of the automaton as the acceptance condition for $\mathcal{T} \times \mathcal{A}$.

In particular, we can perform this operation if $\mathcal{T} = \mathcal{B}$ is an automaton. We obtain in this way a new automaton $\mathcal{B} \times \mathcal{A}$ that uses the acceptance condition of \mathcal{A} .

We refer the reader to figure 11 for an example on the product of two automata.

► **Proposition 2.3** (Folklore). *Let $\mathcal{B} = (B, \Sigma_1, I_0, \Sigma_2, \delta, \text{Acc}_B)$ be an [automaton](#), and $\mathcal{A} = (A, \Sigma_2, q'_0, \Gamma, \delta', \text{Acc}_A)$ be a [deterministic automaton](#) recognizing $\mathcal{L}(\mathcal{A}) = \text{Acc}_B \subseteq \Sigma_2^\omega$. Then $\mathcal{L}(\mathcal{B} \times \mathcal{A}) = \mathcal{L}(\mathcal{B})$.*

2.4 Classes of acceptance conditions

The definition of an [acceptance condition](#) we have used so far is a very general one. In fact, a subset $\text{Acc} \subseteq \Gamma^\omega$ can even be non-computable. In this section we present the principal types of representations for the commonly named [\$\omega\$ -regular conditions](#).

Let Γ be a finite set (whose elements will be called *colours*). The set Γ will usually be the set of edges of a [transition system](#).

Büchi A [Büchi condition](#) Acc_B is represented by a subset $B \subseteq \Gamma$. An infinite word $u \in \Gamma^\omega$ is accepted if some colour from B appears infinitely often in u :

$$u \in \text{Acc}_B \Leftrightarrow \text{Inf}(u) \cap B \neq \emptyset.$$

The dual notion is the co-Büchi condition.

co-Büchi The [co-Büchi condition](#) Acc_{cB} associated to $B \subseteq \Gamma$ is defined as

$$u \in \text{Acc}_{cB} \Leftrightarrow \text{Inf}(u) \cap B = \emptyset.$$

Rabin A [Rabin condition](#) is represented by a family of “Rabin pairs”, $R = \{(E_1, F_1), \dots, (E_r, F_r)\}$, where $E_i, F_i \subseteq \Gamma$. The condition Acc_R is defined as

$$u \in \text{Acc}_R \Leftrightarrow \text{there is an index } i \in \{1, \dots, r\} \text{ such that } \text{Inf}(u) \cap E_i \neq \emptyset \wedge \text{Inf}(u) \cap F_i = \emptyset.$$

The dual notion of a Rabin condition is the Streett condition.

Streett The *Streett condition* associated to the family $S = \{(E_1, F_1), \dots, (E_r, F_r)\}$, $E_i, F_i \subseteq \Gamma$ is defined as

$$u \in Acc_S \Leftrightarrow \text{for all } i \in \{1, \dots, r\} \quad Inf(u) \cap E_i \neq \emptyset \rightarrow Inf(u) \cap F_i \neq \emptyset.$$

Parity To define a *parity condition* we suppose that Γ is a finite subset of \mathbb{N} . We define the condition Acc_P as

$$u \in Acc_P \Leftrightarrow \min Inf(u) \text{ is even.}$$

The elements of Γ are called *priorities* in this case. Since the expressive power of a parity condition (and the complexity of related algorithms) depends on the number of priorities used, we associate to a parity condition the interval $[\mu, \eta]$, where $\mu = \min \Gamma$ and $\eta = \max \Gamma$. Modulo a normalization (subtracting μ or $\mu - 1$ to all priorities) we can suppose that $\mu = 0$ or $\mu = 1$. If a parity condition uses priorities in $[\mu, \eta]$ we call it a *$[\mu, \eta]$ -parity condition*.

We remark that *Büchi conditions* are exactly $[0, 1]$ -parity conditions and *co-Büchi* are $[1, 2]$ -parity conditions.

Parity conditions are also called *Rabin chain conditions* since a parity condition is equivalent to a Rabin condition given by $R = \{(E_1, F_1), \dots, (E_r, F_r)\}$ with $F_1 \subseteq E_1 \subseteq F_2 \subseteq \dots \subseteq E_r$.

Muller A *Muller condition* is given by a family $\mathcal{F} \subseteq \mathcal{P}(\Gamma)$ of subsets of Γ . A word $u \in \Gamma^\omega$ is accepted if the colours visited infinitely often form a set of the family \mathcal{F} .

$$u \in Acc_{\mathcal{F}} \Leftrightarrow Inf(u) \in \mathcal{F}.$$

We remark that Muller conditions can express all the previously defined acceptance conditions.

We will also define conditions that depend on the structure of the transition system and not only on the set of colours.

Generalized weak conditions Let $\mathcal{T} = (V, E, Source, Target, q_0, Acc)$ be a *transition system*.

An *ordered partition* of \mathcal{T} is a partition of V , $V_1, \dots, V_s \subseteq V$ such that for every pair of vertices $p \in V_i$, $q \in V_j$, if there is a transition from p to q , then $i \geq j$. We call each subgraph V_i a *component of the ordered partition*. Every such component must be a union of *strongly connected components* of \mathcal{T} , so we can imagine that the partition is the decomposition into strongly connected components suitably ordered. We remark that given an *ordered partition* of \mathcal{T} , a *run* will eventually stay in some component V_i .

Given different representations of acceptance conditions Acc_1, \dots, Acc_m from some of the previous classes, a *generalised weak condition* is a condition for which we allow to use the different conditions in different components of an ordered partition of a transition system.

We will mainly use the following type of *generalised weak* condition:

Given a transition system \mathcal{T} and an *ordered partition* $(V_i)_{i=1}^s$, a *Weak_k*-condition is a *parity condition* such that in any component V_i there are at most k different priorities associated to transitions between vertices in V_i . It is the *generalised weak condition* for $[1, k]$ and $[0, k - 1]$.

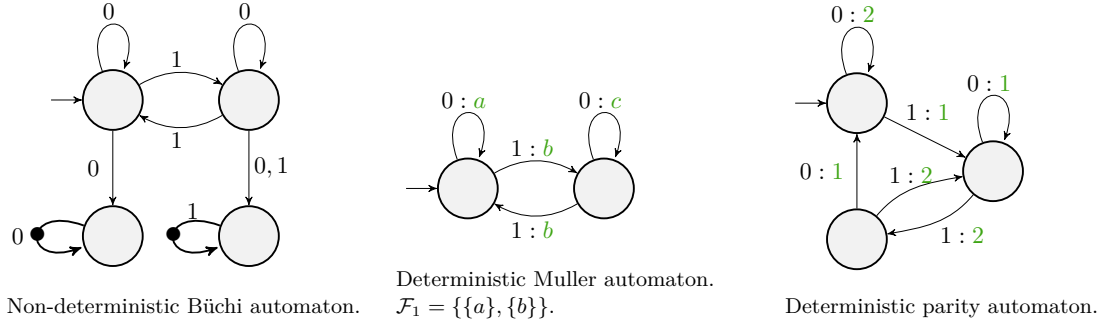
The adjective *Weak* has typically been used to refer to the condition *Weak₁*. It correspond to a partition of \mathcal{T} into “accepting” and “rejecting” components. A *run* will be accepted if the component it finally stays in is accepting.

Transition systems (resp. automata, games) using an acceptance condition of type \mathcal{R} will be called \mathcal{R} -transition systems (resp. \mathcal{R} -automata, \mathcal{R} -games). We will also say that they are labelled with an \mathcal{R} -condition.

► **Example 2.4.** In figure 1 we show three automata recognizing the language

$$\mathcal{L} = \{u \in \{0,1\}^\omega : \text{Inf}(u) = \{1\} \text{ or } (\text{Inf}(u) = \{0\} \text{ and there is an even number of 1's in } u)\}$$

and using different classes of acceptance conditions. We represent Büchi conditions by marking the accepting transitions with a \bullet symbol. For Muller or parity conditions we write in each transition $\alpha : a$, with $\alpha \in \{0,1\}$ the input letter and $a \in \Gamma$ the output letter. The initial vertices are represented with an incoming arrow.



■ **Figure 1** Different types of automata accepting the language \mathcal{L} .

In the following we will use a small abuse of notation and speak indifferently of an acceptance condition and its representation. For example, we will sometimes replace the acceptance condition of a transition system by a family of sets \mathcal{F} (representing a Muller condition) or by a function assigning priorities to edges.

Equivalent conditions

Two different representations of acceptance conditions over a set Γ are *equivalent* if they define the same set $\text{Acc} \subseteq \Gamma^\omega$.

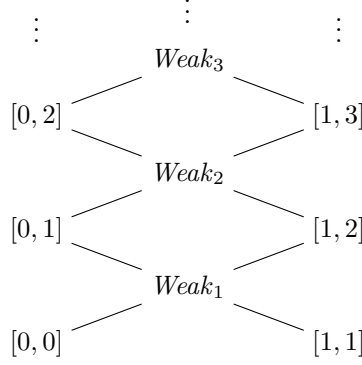
Given a transition system graph \mathcal{T}_G , two representations $\mathcal{R}_1, \mathcal{R}_2$ of acceptance conditions are *equivalent over* \mathcal{T}_G if they define the same accepting subset of runs of $\text{Run}_{\mathcal{T}_G}$. We write $(\mathcal{T}_G, \mathcal{R}_1) \simeq (\mathcal{T}_G, \mathcal{R}_2)$.

The deterministic parity hierarchy

As we have mentioned in the introduction, deterministic Büchi automata have strictly less expressive power than deterministic Muller automata. However, every language recognized by a Muller automaton can be recognized by a deterministic parity automaton, but we might require more or less priorities to do so. We can assign to each regular language $L \subseteq \Sigma^\omega$ the optimal number of priorities needed to recognise it using a deterministic automaton. We obtain in this way the *deterministic parity hierarchy*, first introduced by Mostowski in [17], represented in figure 2. In that figure, we denote by $[\mu, \eta]$ the set of languages over an alphabet Σ that can be recognized using a deterministic $[\mu, \eta]$ -parity automaton. The

intersection of the levels $[0, k]$ and $[1, k + 1]$ is exactly the set of languages recognized using a $Weak_k$ deterministic automaton.

This hierarchy is strict, that is, for each level of the hierarchy there are languages that do not appear in lower levels [26].



■ **Figure 2** The deterministic parity hierarchy.

We observe that the set of languages that can be recognised by a deterministic **Rabin** automaton using r Rabin pairs is the level $[1, 2r + 1]$. Similarly, the languages recognisable by a deterministic **Streett** automaton using s pairs is $[0, 2s]$.

For non-deterministic automata the hierarchy collapses for the level $[0, 1]$ (Büchi automata).

2.5 Trees

A **tree** is a set of sequences of non-negative integers $T \subseteq \omega^*$ that verifies:

- T is prefix-closed: if $\tau \cdot i \in T$, for $\tau \in \omega^*, i \in \omega$, then $\tau \in T$.
- T is order-closed: if $\tau \cdot i \in T$, for $\tau \in \omega^*, i \in \omega$, then $\tau \cdot j \in T$ for all $j < i$.

In this report we will only consider finite trees.

The elements of T are called **nodes**. The empty sequence ε belongs to every non-empty tree and it is called the **root** of the tree. A **node** is called a **leaf** of T if it is a maximal sequence of T (for the **prefix relation** \sqsubseteq). A **branch** of T is the set of prefixes of a **leaf**. The set of branches of T is denoted $Branch(T)$. A **node** of the form $\tau \cdot i$, $i \in \omega$, is called a **child** of τ , and τ is called its **parent**. We let $Children(\tau)$ denote the set of children of a node τ . Two different children σ_1, σ_2 of τ are called **siblings**, and we say that σ_1 is **older** than σ_2 if $Last(\sigma_1) < Last(\sigma_2)$. If two **nodes** τ, σ verify $\tau \sqsubseteq \sigma$, then τ is called an **ancestor** of σ , and σ a **descendant** of τ (we add the adjective “strict” if in addition they are not equal).

For a node $\tau \in T$ we define $Subtree(\tau)$ as the set of nodes that appear below τ , or above it in the same branch (they are **ancestors** or **descendants** of τ): $Subtree(\tau) = \{\sigma \in T : \sigma \sqsubseteq \tau \text{ or } \tau \sqsubseteq \sigma\}$.

We will sometimes need to choose a maximal node for \sqsubseteq in a subtree of T in a deterministic way. We will pick the leftmost leaf of that subtree. In order to formalise it, we refine the order \sqsubseteq into a total order \preceq taking a modified lexicographic order:

$$\tau \preceq \sigma \Leftrightarrow \tau \sqsubseteq \sigma \text{ or } \exists i < |\tau| \text{ such that } \sigma(i) < \tau(i) \text{ and } \tau(j) = \sigma(j) \text{ for } j < i$$

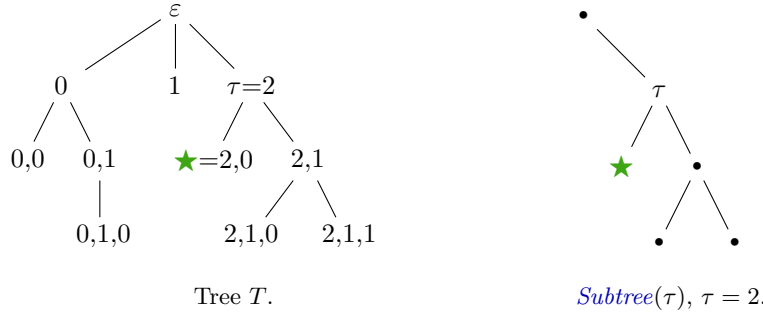
Given a node τ of a tree T , the *depth* of τ in T , is defined as the length of τ , $Depth(\tau) = |\tau|$ (the root ε has depth 0). The *height of a tree* T , written $Height(T)$, is defined as the maximal length of a *leaf* of T plus 1. The *height of the node* $\tau \in T$ is $Height(T) - Depth(\tau)$ (maximal leaves have height 1).

A *labelled tree* is a pair (T, ν) , where T is a *tree* and $\nu : T \rightarrow \Lambda$ is a labelling function into a set of labels Λ .

We write sequences of numbers separating them by commas.

► **Example 2.5.** In figure 3 we show a tree T of *height* 4 and we show $Subtree(\tau)$ for $\tau = 2$. The node τ has *depth* 1 and height 3.

The node $\star = 2,0$ is maximal for the relation \sqsubseteq in $Subtree(\tau)$. It is also the oldest *child* of the node $\tau = 2$.



■ **Figure 3** Example of a tree.

3 An optimal transformation of Muller into parity conditions

In the previous section we have presented different classes of *acceptance conditions* for *transition systems* over infinite words, with *Muller conditions* being the most general kind of *ω -regular conditions*. In order to translate a *Muller condition* \mathcal{F} over Γ into a simpler one, the usual procedure is to build a *deterministic automaton* over Γ using a simpler condition that accepts \mathcal{F} , i.e., this automaton will accept the words $u \in \Gamma^\omega$ such that $Inf(u) \in \mathcal{F}$. As we have asserted, the simplest condition that we could use in general in such a deterministic automaton is a *parity* one, and the number of priorities that we can use is determined by the position of the *Muller condition* in the *parity hierarchy*.

In this section we build a parity automaton that recognises a given *Muller condition*, and we prove that this automaton has minimal size and uses the optimal number of priorities. This construction is based in the notion of the *Zielonka tree*, introduced in [27] (applied there to the study of the optimal memory needed to solve a Muller game). In most cases, this automaton strictly improves other constructions such as the LAR [9] or its modifications [13].

All constructions and proofs of this section can be regarded as a special case of those of section 4. However, we include the proofs for this case here since we think that this will help the reader to understand many ideas that will reappear in section 4 in a more complex context.

3.1 The Zielonka tree automaton

In this first section we present the Zielonka tree and the parity automaton that it induces.

► **Definition 3.1** (Zielonka tree of a Muller condition). Let Γ be a finite set of colours and $\mathcal{F} \subseteq \mathcal{P}(\Gamma)$ a *Muller condition* over Γ . The *Zielonka tree* of \mathcal{F} , written $T_{\mathcal{F}}$, is a tree labelled with subsets of Γ via the labelling $\nu : T_{\mathcal{F}} \rightarrow \mathcal{P}(\Gamma)$, defined inductively as:

- $\nu(\varepsilon) = \Gamma$
- If τ is a node already constructed labelled with $S = \nu(\tau)$, we let S_1, \dots, S_k be the maximal subsets of S verifying the property

$$S_i \in \mathcal{F} \Leftrightarrow S \notin \mathcal{F} \quad \text{for each } i \in [1, k].$$

For each $i = 1, \dots, k$ we add a child to τ labelled with S_i .

► **Remark.** We have not specified the order in which children of a node appear in the Zielonka tree. Therefore, strictly speaking there will be several Zielonka trees of a Muller condition. The order of the nodes will not have any relevance in this work and we will speak of “the” Zielonka tree of \mathcal{F} .

► **Definition 3.2.** We say that the condition \mathcal{F} and the tree $T_{\mathcal{F}}$ are *even* if $\Gamma \in \mathcal{F}$, and that they are *odd* on the contrary.

We associate a priority $p_Z(\tau)$ to each node (to each level in fact) of the Zielonka tree as follows:

- If $T_{\mathcal{F}}$ is *even*, then $p_Z(\tau) = \text{Depth}(\tau)$.
- If $T_{\mathcal{F}}$ is *odd*, then $p_Z(\tau) = \text{Depth}(\tau) + 1$.

In this way, $p_Z(\tau)$ is even if and only if $\nu(\tau) \in \mathcal{F}$. We represent nodes $\tau \in T_{\mathcal{F}}$ such that $p_Z(\tau)$ is even as a *circle* (round nodes), and those for which $p_Z(\tau)$ is odd as a *square*.

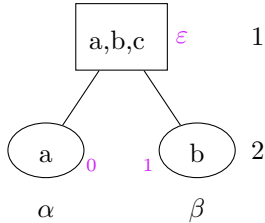
► **Example 3.3.** Let $\Gamma_1 = \{a, b, c\}$ and $\mathcal{F}_1 = \{\{a\}, \{b\}\}$ (the Muller condition of the automaton of example 2.4). The Zielonka tree $T_{\mathcal{F}_1}$ is shown in figure 4. It is *odd*.

Let $\Gamma_2 = \{a, b, c, d\}$ and

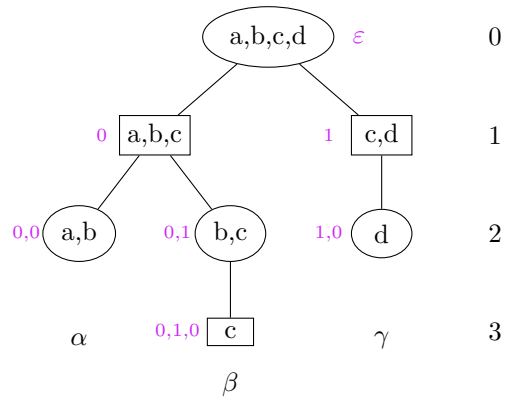
$$\mathcal{F}_2 = \{\{a, b, c, d\}, \{a, b, d\}, \{a, c, d\}, \{b, c, d\}, \{a, b\}, \{a, d\}, \{b, c\}, \{b, d\}, \{a\}, \{b\}, \{d\}\}.$$

The Zielonka tree $T_{\mathcal{F}_2}$ is *even* and it is shown on figure 5.

On the right of each tree there are the priorities assigned to the nodes of the corresponding level. We have named the branches of the Zielonka trees with greek letters and we indicate the names of the nodes in *violet*.



■ **Figure 4** Zielonka tree $T_{\mathcal{F}_1}$.



■ **Figure 5** Zielonka tree $T_{\mathcal{F}_2}$.

We show next how to use the [Zielonka tree](#) of \mathcal{F} to build a [deterministic automaton](#) recognizing the [Muller condition](#) \mathcal{F} . We start by giving some technical definitions.

► **Definition 3.4.** For a branch $\beta \in \text{Branch}(T_{\mathcal{F}})$ and a colour $a \in \Gamma$ we define $\text{Supp}(\beta, a) = \tau$ as the [deepest](#) node (maximal for \sqsubseteq) in β such that $a \in \nu(\tau)$.

For a node τ and a colour $a \in \Gamma$ we let $\text{Below}(\tau, a)$ be the maximal node for \sqsubseteq (the leftmost node) in $\text{Subtree}(\tau)$ such that $a \in \nu(\tau)$. We define $\text{Leftbranch}(\tau, a)$ as the branch $\beta \in \text{Branch}(T_{\mathcal{F}})$ defined by a leaf σ such that σ is maximal for \sqsubseteq in $\text{Subtree}(\text{Below}(\tau, a))$.

Intuitively, if $a \in \nu(\tau)$, then $\text{Leftbranch}(\tau, a)$ is the branch defined by the following process: we start at τ , and we descend at steps taking the leftmost child that contains the colour a . If at some step there is no child containing a , we take the leftmost child in the following steps until arriving to a leaf, that will define this branch.

► **Example 3.5.** In the previous example, on the tree $T_{\mathcal{F}_2}$ of figure 5, we have that $\text{Supp}(\alpha, c) = 0$ (node labelled with $\{a, b, c\}$), $\text{Below}(0, c) = 0, 1, 0$ and $\text{Leftbranch}(0, c) = \beta$.

► **Definition 3.6** (Zielonka tree automaton). Given a [Muller condition](#) \mathcal{F} over Γ with [Zielonka tree](#) $T_{\mathcal{F}}$, we define the [Zielonka tree automaton](#) $\mathcal{Z}_{\mathcal{F}} = (Q, \Gamma, q_0, [\mu, \eta], \delta, p : Q \times \Gamma \rightarrow [\mu, \eta])$ as

- $Q = \text{Branch}(T_{\mathcal{F}})$, the set of states is the set of branches of $T_{\mathcal{F}}$.
- The initial state q_0 is irrelevant, we pick the leftmost branch.
- $\delta(\beta, a) = \text{Leftbranch}(\text{Supp}(\beta, a), a)$.
- $\mu = 0, \eta = \text{Height}(T_{\mathcal{F}}) - 1$ if \mathcal{F} is [even](#) and $\mu = 1, \eta = \text{Height}(T_{\mathcal{F}})$ if \mathcal{F} is [odd](#).
- $p(\beta, a) = p_Z(\text{Supp}(\beta, a), a)$.

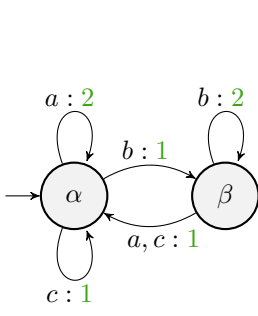
Intuitively, to determine the [run](#) produced by a word $w \in \Gamma^\omega$ we can imagine that we move through the nodes of $T_{\mathcal{F}}$ and the state of the [Zielonka tree automaton](#) will be a branch containing this node. If we are placed in a node τ , the leftmost branch β under τ containing the colour we have just read indicates the state of the automaton we are in. If we are in the branch β and read a colour $a \in \Gamma$, we move to the deepest node in β that contains this colour, and we produce as a priority the [depth](#) of this node. The state in $\mathcal{Z}_{\mathcal{F}}$ will be the leftmost branch under this new node containing a (maybe none of the children contains a , we just pick the leftmost branch).

► **Example 3.7.** Let us consider the conditions of example 3.3. The [Zielonka tree automaton](#) for the [Muller condition](#) \mathcal{F}_1 is shown in figure 6, and that for \mathcal{F}_2 in figure 7. States are the branches of the respective [Zielonka tree](#).

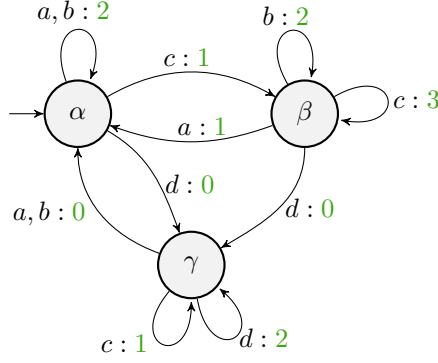
► **Proposition 3.8** (Correctness). Let $\mathcal{F} \subseteq \mathcal{P}(\Gamma)$ be a [Muller condition](#) over Γ . Then, a word $u \in \Gamma^\omega$ verifies $\text{Inf}(u) \in \mathcal{F}$ (u belongs to the Muller condition) if and only if u is [accepted by](#) $\mathcal{Z}_{\mathcal{F}}$.

Proof. Let us suppose that $\text{Inf}(u) \in \mathcal{F}$. Since we can start from an arbitrary state of $\mathcal{Z}_{\mathcal{F}}$, we can suppose that all letters appearing in u are those in $\text{Inf}(u)$. We claim that there is a node in $T_{\mathcal{F}}$, that we will call τ , such that $\text{Inf}(u) \subseteq \nu(\tau)$, that every child σ of τ verifies $\text{Inf}(u) \not\subseteq \nu(\sigma)$ and such that the [run over](#) u in $\mathcal{Z}_{\mathcal{F}}$ visits only branches in $\text{Subtree}(\tau)$ and it takes infinitely often transitions corresponding to the node τ (intuitively, we stay always in the nodes below τ , but infinitely often we have to go up to τ and produce $p_Z(\tau)$). Since τ is a maximal node containing $\text{Inf}(u)$ and $\text{Inf}(u) \in \mathcal{F}$, the priority $p_Z(\tau)$ is even and it is the least priority produced.

Indeed, take τ_1 as the deepest node in the initial branch that contains all colours of $\text{Inf}(u)$, and τ the maximal node for \sqsubseteq in $\text{Subtree}(\tau_1)$ containing all colours of $\text{Inf}(u)$. This



■ **Figure 6** The Zielonka tree automaton $\mathcal{Z}_{\mathcal{F}_1}$.



■ **Figure 7** The Zielonka tree automaton $\mathcal{Z}_{\mathcal{F}_2}$.

claim holds because each time that we read all letters in $\text{Inf}(u)$ we have to ascend to τ , but as we never read other colours we never go higher nor visit branches not in $\text{Subtree}(\tau)$.

The reasoning is symmetric if $\text{Inf}(u) \notin \mathcal{F}$. ◀

3.2 Optimality of the Zielonka tree automaton

We prove in this section the optimality of the Zielonka tree automaton, both for number of priorities (proposition 3.9) and for size (theorem 3.13).

The next proposition can be proved easily applying the results of [19]. We present here a self-contained proof.

► **Proposition 3.9** (Optimal number of priorities). *The Zielonka tree gives the optimal number of priorities recognizing a Muller condition \mathcal{F} . More precisely, if $[\mu, \eta]$ are the priorities used by $\mathcal{Z}_{\mathcal{F}}$ and \mathcal{P} is another parity automaton recognizing \mathcal{F} that uses priorities in $[\mu', \eta']$, then $\eta - \mu \leq \eta' - \mu'$, and if they are equal, μ and μ' have the same parity.*

Proof. Let \mathcal{P} be a deterministic parity automaton recognizing \mathcal{F} . Let b be a branch of $\mathcal{T}_{\mathcal{F}}$ of maximal length h , and let $S_0 \subseteq S_2 \subseteq \dots \subseteq S_{h-1} = \Gamma$ be the labellings of the nodes of this branch from bottom to top. Let us suppose $S_0 \in \mathcal{F}$, being the case $S_0 \notin \mathcal{F}$ symmetric. Let a_i be the finite word formed concatenating the colours of S_i . In particular a_i is accepted iff i is even. Let η' be the greatest priority appearing in the automaton \mathcal{P} . We prove by recursion on j that, for every $v \in \Gamma^*$, the run over $(a_0 a_1 \dots a_j v)^\omega$ in \mathcal{P} produces a priority smaller than or equal to $\eta' - j$, if η' even, and smaller than or equal to $\eta' - j - 1$ if η' is odd. We do here the case η' even, being the case η' odd symmetric.

For $j = 0$ this is clear, since η' is the greatest priority. For $j > 0$, if it was not true, the smallest priority produced infinitely often reading $(a_0 a_1 \dots a_j v)^\omega$ would be strictly greater than $\eta' - j$ for some $v \in \Gamma^*$. Since $\eta' - j$ has the same parity than j and $S_j \in \mathcal{F}$ if and only if j is even, then the smallest priority produced infinitely often reading $(a_0 a_1 \dots a_j v)^\omega$ must have the same parity than j and cannot be $\eta' - j + 1$, so it is greater than $\eta' - j + 2$. However, by recursion hypothesis, the run over $(a_0 a_1 \dots a_{j-1} w)^\omega$ produces a priority smaller than or equal to $\eta' - (j - 1)$ for every w , in particular for $w = a_j v$, contradicting the recursion hypothesis.

In particular, taking $v = \varepsilon$, we have proved that the run over $(a_0 a_1 \dots a_{h-1})^\omega$ in \mathcal{P} produces a priority smaller than or equal to $\eta' - (h - 1)$, and \mathcal{P} must use priorities in $[\eta' - (h - 1), \eta']$, that is, h priorities.

◀

In order to prove theorem 3.13 we introduce the definition of an X -strongly connected component and we present two key lemmas.

► **Definition 3.10.** Let $\mathcal{A} = (Q, \Sigma, q_0, \Gamma, \delta, \text{Acc})$ be a *deterministic automaton* and $X \subseteq \Sigma$ a subset of letters of the input alphabet. An X -strongly connected component (abbreviated X -SCC) is a non-empty subset of states $S \subseteq Q$ such that:

- For every $q \in S$ and every $x \in X$, $\delta(q, x) \in S$.
- For every pair of states $q, q' \in S$ there is a finite word $w \in X^*$ such that $\delta(q, w) = q'$.

That is, an X -SCC of \mathcal{A} are the states of a X -complete part of \mathcal{A} that forms a *strongly connected subgraph*.

► **Lemma 3.11.** For every *deterministic automaton* $\mathcal{A} = (Q, \Sigma, q_0, \Gamma, \delta, \text{Acc})$ and every subset of letters $X \subseteq \Sigma$ there is an *accessible X -SCC* in \mathcal{A} .

Proof. Restricting ourselves to the set of *accessible* states of \mathcal{A} we can suppose that every state of the automaton is accessible.

We prove the lemma by induction on $|\mathcal{A}|$. For $|\mathcal{A}| = 1$, the state of the automaton forms an X -SCC. For $|\mathcal{A}| > 1$, if Q is not an X -SCC, there are $q, q' \in Q$ such that there does not exist a word $w \in X^*$ such that $\delta(q, w) = q'$. Let

$$Q_q = \{p \in Q : \exists u \in X^* \text{ such that } p = \delta(q, u)\}.$$

Since $q' \notin Q_q$, the set Q_q is strictly smaller than Q . The set Q_q is non-empty and closed under transitions labelled by letters of X , so the restriction of \mathcal{A} to this set of states and the alphabet X forms an automaton $\mathcal{A}_{Q_q, X} = (Q_q, X, q, \Gamma, \delta')$ (where δ' is the restriction of δ to these states and letters). By induction hypothesis, $\mathcal{A}_{Q_q, X}$ contains an X -SCC that is also an X -SCC for \mathcal{A} . ◀

► **Lemma 3.12.** Let \mathcal{F} be a *Muller condition* over Γ , $T_{\mathcal{F}}$ its *Zielonka tree* and $\mathcal{P} = (P, \Gamma, p_0, [\mu', \eta'], \delta_P, p' : P \rightarrow [\mu', \eta'])$ a *deterministic parity automaton* recognizing \mathcal{F} . Let τ be a node of $T_{\mathcal{F}}$ and $C = \nu(\tau) \subseteq \Gamma$ its label. Finally, let $A, B \subseteq C$ be the labels of two different *children* of τ , i.e. they appear in \mathcal{F} if and only if C does not, and they are maximal for this “alternating property”. Then, if P_A and P_B are two *accessible A -SCC* and *B -SCC* of \mathcal{P} respectively, they satisfy $P_A \cap P_B = \emptyset$.

Proof. We can suppose that $C \in \mathcal{F}$ and $A, B \notin \mathcal{F}$. Suppose that there is a state $q \in P_A \cap P_B$. Let $\{a_1, \dots, a_l\} = A$, $\{b_1, \dots, b_r\} = B$ and $q_1 = \delta(q, a_1 \dots a_l) \in A$, $q_2 = \delta(q, b_1 \dots b_r) \in B$. By definition of an X -SCC, there are words $u_1 \in A^*$, $u_2 \in B^*$ such that $\delta(q_1, u_1) = q$, $\delta(q_2, u_2) = q$. As $A, B \notin \mathcal{F}$, the maximal priorities p_1 and p_2 produced by the *runs over* $(a_1 \dots a_l u_1)^\omega$ and $(b_1 \dots b_r u_2)^\omega$ starting from q will be odd. However, the run over $(a_1 \dots a_l u_1 b_1 \dots b_r u_2)^\omega$ starting from q must produce an even maximal priority (as $A \cup B \in \mathcal{F}$), but the maximal priority visited in this run is $\max\{p_1, p_2\}$, odd, which leads to a contradiction. ◀

► **Theorem 3.13** (Optimal size of the Zielonka tree automaton). Every *deterministic parity automaton* $\mathcal{P} = (P, \Gamma, p_0, [\mu', \eta'], \delta_P, p' : P \times \Gamma \rightarrow [\mu', \eta'])$ accepting a *Muller condition* \mathcal{F} over Γ verifies

$$|\mathcal{Z}_{\mathcal{F}}| \leq |\mathcal{P}|.$$



Proof. Let \mathcal{P} be a deterministic parity automaton accepting \mathcal{F} . To show $|\mathcal{Z}_{\mathcal{F}}| \leq |\mathcal{P}|$ we proceed by induction on the number of colours $|\Gamma|$. For $|\Gamma| = 1$ the two possible [Zielonka tree automata](#) have one state, so the result holds. Suppose $|\Gamma| > 1$ and consider the first level of $\mathcal{T}_{\mathcal{F}}$. Let n be the number of children of the root of $\mathcal{T}_{\mathcal{F}}$. For $i = 1, \dots, n$ let $A_i = \nu(\tau_i) \subseteq C$ be the label of the i -th child of the root of $\mathcal{T}_{\mathcal{F}}$, τ_i , and let n_i be the number of branches of the subtree under τ_i , $\text{Subtree}(\tau_i)$. We remark that $|\mathcal{Z}_{\mathcal{F}}| = \sum_{i=1}^n n_i$. Let $\mathcal{F} \upharpoonright A_i := \{F \in \mathcal{F} : F \subseteq A_i\}$. Since each A_i verifies $|A_i| < |C|$ and the [Zielonka tree](#) for $\mathcal{F} \upharpoonright A_i$ is the subtree of $\mathcal{T}_{\mathcal{F}}$ under the node τ_i , every [deterministic parity automaton](#) accepting $\mathcal{F} \upharpoonright A_i$ has at least n_i states, by induction hypothesis.

After lemma 3.11, for each $i = 1, \dots, n$ there is an accessible A_i -SCC in \mathcal{P} , called P_i . Therefore, the restriction to P_i (with a random initial state) is an automaton recognising $F \upharpoonright A_i$. By induction hypothesis, for each $i = 1, \dots, n$, $|P_i| \geq n_i$. After lemma 3.12, we know that for every $i, j \in \{1, \dots, n\}$, $i \neq j$, $P_i \cap P_j = \emptyset$. We deduce that

$$|\mathcal{P}| \geq \sum_{i=1}^n |P_i| \geq \sum_{i=1}^n n_i = |\mathcal{Z}_{\mathcal{F}}|.$$

◀

3.3 The Zielonka tree of some classes of acceptance conditions

As we have seen in section 2.4, [Muller conditions](#) is a broad class of [acceptance conditions](#) that can express [parity](#), [Rabin](#) or [Streett](#) conditions. The [Zielonka tree](#) of a Muller condition discloses its structure, and we show in this section how we can use the Zielonka tree to deduce if it is representable by a condition of a different class.

► **Proposition 3.14.** *A Muller condition $\mathcal{F} \subseteq \mathcal{P}(\Gamma)$ is equivalent to a parity condition $p : \Gamma \rightarrow \mathbb{N}$ if and only if the Zielonka tree $T_{\mathcal{F}}$ has a single branch.*

Moreover, \mathcal{F} is equivalent to a $[1, \eta]$ -parity condition (resp. $[0, \eta - 1]$ -parity condition) if and only if $T_{\mathcal{F}}$ has a single branch and $\text{Height}(T_{\mathcal{F}}) \leq \eta$, and in case of equality $T_{\mathcal{F}}$ is odd (resp. even).

Proof. We do the proof for the case \mathcal{F} odd.

If $T_{\mathcal{F}}$ has a single branch of length η , we can define over Γ a parity condition as follows: we assign to the elements of the label of the deepest node of $T_{\mathcal{F}}$ the priority η . From there, we go up the tree, step by step, and we assign to the new colours appearing in each level a priority of one unity less than in the previous step (so in the end the elements that only appear in the root will be assigned the priority 1).

Conversely, suppose that we can assign priorities to the elements of Γ by $p : \Gamma \rightarrow [1, \eta]$, obtaining an equivalent condition for \mathcal{F} . We show that any node of the Zielonka tree $T_{\mathcal{F}}$ has at most one child. Indeed, suppose that $\tau \in T_{\mathcal{F}}$ and that the least element in its label $A = \nu(\tau)$ has odd priority p_1 (symmetric if it is even). Let p_2 be the smallest even priority in A . In every child of τ the elements with a smaller priority than p_2 must disappear, so the set of elements $A \cap \{c \in \Gamma : p(c) \geq p_2\}$ is the only maximal subset of A belonging to \mathcal{F} . Moreover, in the label of the child of τ there is at least one priority less, so the height of $T_{\mathcal{F}}$ will be at most η . ◀

► **Proposition 3.15.** *A Muller condition $\mathcal{F} \subseteq \mathcal{P}(\Gamma)$ is equivalent to a Rabin condition R if and only if every round node of $T_{\mathcal{F}}$ (nodes with an even priority) has at most one child.*

The condition \mathcal{F} is equivalent to a Streett condition S if and only if every square node of $T_{\mathcal{F}}$ (nodes with an odd priority) has at most one child.

Proof. We prove it for the Rabin case, being Streett conditions the dual notion.

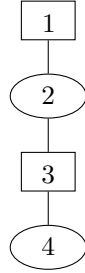
If all **round nodes** of $T_{\mathcal{F}}$ have at most one child we define a **Rabin condition** equivalent to \mathcal{F} as follows: for each child σ of a **square node** τ we add a Rabin pair (E_{σ}, F_{σ}) with $F_{\sigma} = \Gamma \setminus \nu(\sigma)$. In order to define E_{σ} we observe that σ has a single child σ' (if σ is a leaf we set $\nu(\sigma') = \emptyset$), so we can define $E_{\sigma} = \nu(\sigma) \setminus \nu(\sigma')$. This is, the pair (E_{σ}, F_{σ}) accepts the sets of colours $A \subseteq \Gamma$ that contain some of the colours that disappear in the step $\sigma \rightarrow \sigma'$ and none of the colours appearing up in the tree. It is easy to verify that this condition is equivalent to \mathcal{F} .

Conversely, suppose that \mathcal{F} is **equivalent** to a **Rabin condition** given by the pairs $R = \{(E_1, F_1), \dots, (E_r, F_r)\}$. If $\tau \in T_{\mathcal{F}}$ is a **round node** ($A = \nu(\tau) \in \mathcal{F}$), then its label A contains some colours that belongs to E_{i_1}, \dots, E_{i_k} and none belonging to F_{i_1}, \dots, F_{i_k} for some i_1, \dots, i_k , $k \geq 1$. Any child of τ must not have these colours, so the only maximal subset of A that is not in \mathcal{F} is $A \setminus (E_{i_1} \cup \dots \cup E_{i_k})$.

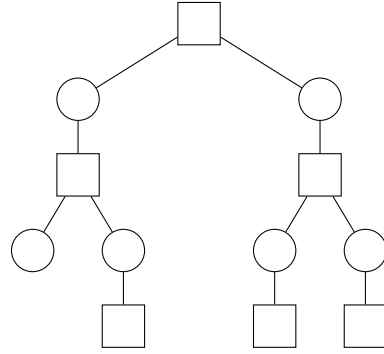
◀

► **Example 3.16.** In figures 8 and 9 we represent **Zielonka trees** for some examples of **parity** and **Rabin** conditions.

We remark that for a fixed number of **Rabin** (or **Streett**) pairs we can obtain **Zielonka trees** of very different shapes that range from a single branch (for **Rabin chain conditions**) to a tree with a branch for each **Rabin pair** and height 3.



■ **Figure 8** Zielonka tree of a parity condition.



■ **Figure 9** Zielonka tree of a Rabin condition.

4 An optimal transformation of Muller into parity transition systems

In this section we present our main contribution: an optimal transformation of **Muller transition systems** into **parity** transition systems. Firstly, we formalise what we mean by “a transformation” using the notion of **locally bijective morphisms** in section 4.1.

Then, we describe a transformation from a **Muller transition system** to a parity one. Most transformations found in the literature use the **product** of the **transition system** by a parity automaton recognising the Muller condition (such as $\mathcal{Z}_{\mathcal{F}}$). In order to achieve optimality this does not suffice, we need to take into account the structure of the transition system. Following ideas already present in [26], we analyse the alternating chains of accepting and rejecting cycles of the transition system. We arrange this information in a collection of **Zielonka trees** obtaining a data structure, the **alternating cycle decomposition**, that subsumes all the structural information of the transition system necessary to decide the acceptance of

a **run**. We present the **alternating cycle decomposition** in 4.2 and we show how to use this structure to obtain a parity transition system that mimics the former Muller one in 4.3.

In section 4.4 we prove the optimality of this construction. More precisely, we prove that if \mathcal{P} is a parity transition system that admits a **locally bijective morphism** to a Muller transition system \mathcal{T} , then the transformation of \mathcal{T} using the alternating cycle decomposition provides a smaller transition system than \mathcal{P} and using less priorities.

4.1 Locally bijective morphisms as witnesses of transformations

We start by defining locally bijective morphisms.

► **Definition 4.1.** Let $\mathcal{T} = (V, E, \text{Source}, \text{Target}, I_0, \text{Acc})$, $\mathcal{T}' = (V', E', \text{Source}', \text{Target}', I'_0, \text{Acc}')$ be two **transition systems**. A **morphism of transition systems**, written $\varphi : \mathcal{T} \rightarrow \mathcal{T}'$, is a pair of maps $(\varphi_V : V \rightarrow V', \varphi_E : E \rightarrow E')$ such that:

- $\varphi_V(v_0) \in I'_0$ for every $v_0 \in I_0$ (initial states are preserved).
- $\text{Source}'(\varphi_E(e)) = \varphi_V(\text{Source}(e))$ for every $e \in E$ (origins of edges are preserved).
- $\text{Target}'(\varphi_E(e)) = \varphi_V(\text{Target}(e))$ for every $e \in E$ (targets of edges are preserved).
- For every **run** $\varrho \in \text{Run}_{\mathcal{T}}$, $\varrho \in \text{Acc} \Leftrightarrow \varphi_E(\varrho) \in \text{Acc}'$ (acceptance condition is preserved).

If (\mathcal{T}, l_V, l_E) , $(\mathcal{T}', l'_V, l'_E)$ are **labelled transition systems**, we say that φ is a **morphism of labelled transition systems** if in addition it verifies

- $l'_V(\varphi_V(v)) = l_V(v)$ for every $v \in V$ (labels of states are preserved).
- $l'_E(\varphi_E(e)) = l_E(e)$ for every $e \in E$ (labels of edges are preserved).

We remark that it follows from the first three conditions that if $\varrho \in \text{Run}_{\mathcal{T}}$ is a **run** in \mathcal{T} , then $\varphi_E(\varrho)$ is a **run** in \mathcal{T}' starting from some initial vertex.

Given a **morphism of transition systems** (φ_V, φ_E) , we will denote both maps by φ whenever no confusion arises. We extend φ_E to E^* and E^ω component wise.

► **Remark.** A **morphism of transition systems** $\varphi = (\varphi_V, \varphi_E)$ is unequivocally characterized by the map φ_E . Nevertheless, it is convenient to keep the notation with both maps.

► **Definition 4.2.** Given two **transition systems** $\mathcal{T} = (V, E, \text{Source}, \text{Target}, q_0, \text{Acc})$, $\mathcal{T}' = (V', E', \text{Source}', \text{Target}', q'_0, \text{Acc}')$, a **morphism of transition systems** $\varphi : \mathcal{T} \rightarrow \mathcal{T}'$ is called

- **Locally surjective** if
 - For every $v'_0 \in I'_0$ there exists $v_0 \in I_0$ such that $\varphi(v_0) = v'_0$.
 - For every $v \in V$ and every $e' \in E'$ such that $\text{Source}'(e') = \varphi(v)$ there exists $e \in E$ such that $\varphi(e) = e'$ and $\text{Source}(e) = v$.
- **Locally injective** if
 - For every $v'_0 \in I'_0$, there is at most one $v_0 \in I_0$ such that $\varphi(v_0) = v'_0$.
 - For every $v \in V$ and every $e' \in E'$ such that $\text{Source}'(e') = \varphi(v)$ if there are $e_1, e_2 \in E$ such that $\varphi(e_i) = e'$ and $\text{Source}(e_i) = v$, for $i = 1, 2$, then $e_1 = e_2$.
- **Locally bijective** if it is both **locally surjective** and **locally injective**.

► **Remark.** Equivalently, a **morphism of transition systems** φ is **locally surjective** (resp. **injective**) if the restriction of φ_E to $\text{Out}(v)$ is a surjection (resp. an injection) into $\text{Out}(\varphi(v))$ for every $v \in V$ and the restriction of φ_V to I_0 is a surjection (resp. an injection) into I'_0 .

If we only consider the **underlying graph** of a **transition system**, without the **accepting condition**, the notion of **locally bijective morphism** is equivalent to the usual notion of bisimulation. However, when considering the accepting condition, we only impose that the acceptance of each **run** must be preserved (and not that the colouring of each transition

is preserved). This allows to compare transition systems using different representations of accepting conditions.

We state two simple, but key facts.

► **Fact 4.3.** *If $\varphi : \mathcal{T} \rightarrow \mathcal{T}'$ is a **locally bijective morphism**, then φ induces a bijection between the runs in $\text{Run}_{\mathcal{T}}$ and $\text{Run}_{\mathcal{T}'}$ that preserves their acceptance.*

► **Fact 4.4.** *If φ is a **locally surjective morphism**, then it is onto the **accessible part** of \mathcal{T}' . In particular if every state of \mathcal{T}' is **accessible**, φ is surjective.*

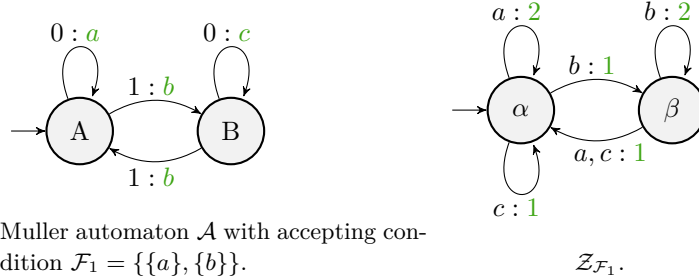
Intuitively, if we transform a **transition system** \mathcal{T}_1 into \mathcal{T}_2 “without adding non-determinism”, we will have a locally bijective morphism $\varphi : \mathcal{T}_2 \rightarrow \mathcal{T}_1$. In particular, if we consider the product of \mathcal{T}_1 by some **deterministic automaton** \mathcal{B} , $\mathcal{T}_2 = \mathcal{T}_1 \times \mathcal{B}$, the projection on the first component gives a **locally bijective morphism** from \mathcal{T}_2 to \mathcal{T}_1 .

► **Example 4.5.** Let \mathcal{A} be the **Muller automaton** presented in the example 1, and $\mathcal{Z}_{\mathcal{F}_1}$ the **Zielonka tree automaton** for its Muller condition $\mathcal{F}_1 = \{\{a\}, \{b\}\}$ as in the figure 6. We show them in figure 10 and their product $\mathcal{A} \times \mathcal{Z}_{\mathcal{F}_1}$ in figure 11.

If we name the states of \mathcal{A} with the letters A and B , and those of $\mathcal{Z}_{\mathcal{F}_1}$ with α, β , there is a locally bijective morphism $\varphi : \mathcal{A} \times \mathcal{Z}_{\mathcal{F}_1} \rightarrow \mathcal{A}$ given by the projection on the first component

$$\varphi_V((X, y)) = X \text{ for } X \in \{A, B\}, y \in \{\alpha, \beta\}$$

and φ_E associates to each edge $e \in \text{Out}(X, y)$ labelled by $a \in \{0, 1\}$ the only edge in $\text{Out}(X)$ labelled with a .



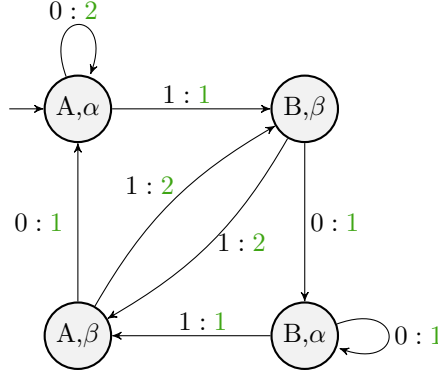
■ **Figure 10** Muller automaton and the Zielonka tree automaton of its acceptance condition.

► **Remark.** We know that $\mathcal{Z}_{\mathcal{F}}$ is a minimal automaton recognizing the **Muller condition** \mathcal{F} (theorem 3.13). However, the product $\mathcal{A} \times \mathcal{Z}_{\mathcal{F}_1}$ has 4 states, and in the example 2.4 (figure 1) we have shown a parity automaton recognizing $\mathcal{L}(\mathcal{A})$ with only 3 states. Moreover, there is a **locally bijective morphism** from this smaller parity automaton to \mathcal{A} (we only have to send the two states on the left to A and the state on the right to B).

In the next section we will show a transformation that will produce the parity automaton with only 3 states starting from \mathcal{A} .

Morphisms of automata and games

Before presenting the optimal transformation of Muller transition systems, we will state some facts about **morphisms** in the particular case of **automata** and **games**. When we speak about a **morphism** between two automata, we always refer implicitly to the morphism between the corresponding **labelled transition systems**, as explained in **example 2.1**.



■ **Figure 11** The product automaton $\mathcal{A} \times \mathcal{Z}_{F1}$.

► **Fact 4.6.** A *morphism* $\varphi = (\varphi_V, \varphi_E)$ between two *deterministic automata* is always *locally bijective* and it is completely characterized by the map φ_V .

Proof. For each letter of the input alphabet and each state, there must be one and only one outgoing transition labelled with this letter. ◀

► **Proposition 4.7.** Let $\mathcal{A} = (Q, \Sigma, I_0, \Gamma, \delta, Acc)$, $\mathcal{A}' = (Q', \Sigma, I'_0, \Gamma, \delta', Acc')$ be two (possibly non-deterministic) *automata*. If there is a *locally surjective morphism* $\varphi : \mathcal{A} \rightarrow \mathcal{A}'$, then $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$.

Proof. Let $u \in \Sigma^\omega$. If $u \in \mathcal{L}(\mathcal{A})$ there is an accepting *run over u* in \mathcal{A} , ϱ . By the definition of a *morphism of labelled transition systems*, $\varphi(\varrho)$ is also an accepting *run over u* in \mathcal{A}' .

Conversely, if $u \in \mathcal{L}(\mathcal{A}')$ there is an accepting *run over u* ϱ' in \mathcal{A}' . Since φ is locally surjective there is a run in \mathcal{A} , ϱ , such that $\varphi(\varrho) = \varrho'$, and therefore ϱ is a run over u and accepting. ◀

► **Remark.** The converse of the previous proposition does not hold: $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$ does not imply the existence of morphisms $\varphi : \mathcal{A} \rightarrow \mathcal{A}'$ or $\varphi : \mathcal{A}' \rightarrow \mathcal{A}$, even if \mathcal{A} has minimal size among the Muller automata recognizing $\mathcal{L}(\mathcal{A})$.

If \mathcal{A} and \mathcal{A}' are *non-deterministic automata* and $\varphi : \mathcal{A} \rightarrow \mathcal{A}'$ is a *locally bijective morphism*, then \mathcal{A} and \mathcal{A}' have to share some other important semantic properties. Two classes of automata that have been extensively studied are unambiguous and good for games automata. An automaton is *unambiguous* if for every input word $w \in \Sigma^\omega$ there is at most one accepting *run over w*. *Good for games* automata (GFG), first introduced by Henzinger and Piterman in [10], are automata that can resolve the non-determinism depending only in the prefix of the word read so far. These types of automata have many good properties and have been used in different contexts (as for example in the model checking of LTL formulas [7] or in the theory of cost functions [5]). Unambiguous automata can recognize ω -regular languages using a *Büchi* condition (see [4]) and GFG automata have strictly more expressive power than deterministic ones, being in some cases exponentially smaller (see [2, 14]).

We omit the proof of the next proposition, being a consequence of fact 4.3 and of the argument from the proof of proposition 4.9.

► **Proposition 4.8.** Let \mathcal{A} and \mathcal{A}' be two *non-deterministic automata*. If $\varphi : \mathcal{A} \rightarrow \mathcal{A}'$ is a *locally bijective morphism*, then

- \mathcal{A} is unambiguous if and only if \mathcal{A}' is unambiguous.
- \mathcal{A} is GFG if and only if \mathcal{A}' is GFG.

Having a [locally bijective morphism](#) between two games implies that the [winning regions](#) of the players are preserved.

► **Proposition 4.9.** *Let $\mathcal{G} = (V, E, \text{Source}, \text{Target}, v_0, \text{Acc}, l_V)$ and $\mathcal{G}' = (V', E', \text{Source}', \text{Target}', v'_0, \text{Acc}', l'_V)$ be two [games](#) such that there is a [locally bijective morphism](#) $\varphi : \mathcal{G} \rightarrow \mathcal{G}'$. Let $P \in \{\text{Eve}, \text{Adam}\}$ be a player in those games. Then, P wins \mathcal{G} if and only if she/he wins \mathcal{G}' . Moreover, if φ is surjective, the [winning region](#) of P in \mathcal{G}' is the image by φ of her/his winning region in \mathcal{G} , $\mathcal{W}_P(\mathcal{G}') = \varphi(\mathcal{W}_P(\mathcal{G}))$.*

Proof. Let $S_P : \text{Run}_{\mathcal{G}} \cap E^* \rightarrow E$ be a winning [strategy](#) for player P in \mathcal{G} . Then, it is easy to verify that the strategy $S'_P : \text{Run}_{\mathcal{G}'} \cap E'^* \rightarrow E'$ defined as $S'_P(\varrho') = \varphi_E(S_P(\varphi^{-1}(\varrho')))$ is a winning [strategy](#) for P in \mathcal{G}' . (Remark that thanks to fact 4.3, the morphism φ induces a bijection over [runs](#), allowing us to use φ^{-1} in this case).

Conversely, if $S'_P : \text{Run}_{\mathcal{G}'} \cap E'^* \rightarrow E'$ is a winning [strategy](#) for P in \mathcal{G}' , then $S_P(\varrho) = \varphi_E^{-1}(S'_P(\varphi(\varrho)))$ is a winning [strategy](#) for P in \mathcal{G} . Here $\varphi_E^{-1}(e')$ is the only edge $e \in E$ in $\text{Out}(\text{Target}(\text{Last}(\varrho)))$ such that $\varphi_E(e) = e'$.

The equality $\mathcal{W}_P(\mathcal{G}') = \varphi(\mathcal{W}_P(\mathcal{G}))$ stems from the fact that if we choose a different initial vertex v_1 in \mathcal{G} , then φ is a [locally bijective morphism](#) to the game \mathcal{G}' with initial vertex $\varphi(v_1)$. Conversely, if we take a different initial vertex v'_1 in \mathcal{G}' , since φ is surjective we can take a vertex $v_1 \in \varphi^{-1}(v'_1)$, and φ remains a locally bijective morphism between the resulting games. ◀

4.2 The alternating cycle decomposition

Most transformations of [Muller](#) into [parity transition systems](#) are based on [products](#) by some automaton converting the Muller condition into a parity one. These transformations act on the totality of the system uniformly, regardless of the local structure of the system and the [acceptance condition](#). However, most [transition systems](#) are not uniform, in the sense that different parts of it might present different configurations: the way in which states are connected as well as the definition of the [acceptance condition](#) differ from one part to another. The transformation we introduce in this section mends this by analysing the interplay between the particular [acceptance condition](#) and the [transition system](#).

► **Definition 4.10.** *Given a [transition system](#) $\mathcal{T} = (V, E, \text{Source}, \text{Target}, q_0, \text{Acc})$, a [loop](#) is a subset of edges $l \subseteq E$ such that it exists $v \in V$ and a finite [run](#) $\varrho \in \text{Run}_{\mathcal{T}, v}$ such that $\text{First}(\varrho) = \text{Last}(\varrho) = v$ and $\text{App}(\varrho) = l$. The set of [loops](#) of \mathcal{T} is denoted $\text{Loop}(\mathcal{T})$. For a [loop](#) $l \in \text{Loop}(\mathcal{T})$ we write*

$$\text{States}(l) := \{v \in V : \exists e \in l, \text{Source}(e) = v\}.$$

Observe that there is a natural partial order in the set $\text{Loop}(\mathcal{T})$ given by set inclusion.

► **Remark.** If l is a [loop](#) in $\text{Loop}(\mathcal{T})$, for every $q \in \text{States}(l)$ there is a run $\varrho \in \text{Run}_{\mathcal{T}, q}$ such that $\text{App}(\varrho) = l$.

► **Remark.** The maximal loops of $\text{Loop}(\mathcal{T})$ (for set inclusion) are disjoint and in one-to-one correspondence with the [strongly connected components](#) of \mathcal{A} .

► **Definition 4.11** (Alternating cycle decomposition). *Let $\mathcal{T} = (V, E, \text{Source}, \text{Target}, q_0, \mathcal{F})$ be a [Muller transition system](#) with [acceptance condition](#) given by $\mathcal{F} \subseteq \mathcal{P}(E)$. The [alternating](#)*

cycle decomposition (abbreviated *ACD*) of \mathcal{T} , noted $\mathcal{ACD}(\mathcal{T})$, is a family of *labelled trees* $(t_1, \nu_1), \dots, (t_r, \nu_r)$ with nodes labelled by *loops* in $\mathcal{Loop}(\mathcal{T})$, $\nu_i : t_i \rightarrow \mathcal{Loop}(\mathcal{T})$. We define it inductively as follows:

- Let $\{l_1, \dots, l_r\}$ be the set of maximal loops of $\mathcal{Loop}(\mathcal{T})$. For each $i \in \{1, \dots, r\}$ we consider a *tree* t_i and define $\nu_i(\varepsilon) = l_i$.
- Given an already defined node τ of a tree t_i we consider the maximal loops of the set

$$\{l \subseteq \nu_i(\tau) : l \in \mathcal{Loop}(\mathcal{T}) \text{ and } l \in \mathcal{F} \Leftrightarrow \nu_i(\tau) \notin \mathcal{F}\}$$

and for each of these loops l we add a child to τ in t_i labelled by l .

For notational convenience we add a special *tree* (t_0, ν_0) with a single node ε labelled with the edges not appearing in any other tree of the forest, i.e., $\nu_0(\varepsilon) = E \setminus \bigcup_{i=1}^r l_i$ (remark that this is not a *loop*).

We define $\text{States}(\nu_0(\varepsilon)) := V \setminus \bigcup_{i=1}^r \text{States}(l_i)$ (remark that this does not follow the general definition of $\text{States}()$ for loops).

We call the trees t_1, \dots, t_r the *proper trees* of the *alternating cycle decomposition* of \mathcal{T} . Given a node τ of t_i , we note $\text{States}_i(\tau) := \text{States}(\nu_i(\tau))$.

► **Remark.** As for the *Zielonka tree*, the *alternating cycle decomposition* of \mathcal{T} is not unique, since it depends on the order in which we introduce the children of each node. This will not affect the upcoming results, and we will refer to it as “the” alternating cycle decomposition of \mathcal{T} .

For the rest of the section we fix a *Muller transition system* $\mathcal{T} = (V, E, \text{Source}, \text{Target}, q_0, \mathcal{F})$ with an *alternating cycle decomposition* given by $(t_1, \nu_1), \dots, (t_r, \nu_r), (t_0, \nu_0)$.

► **Remark.** The *Zielonka tree* for a *Muller condition* \mathcal{F} over the set of colours C can be seen as a special case of this construction, for the automaton with a single state, input alphabet C , a transition for each letter in C and *acceptance condition* \mathcal{F} .

► **Remark.** Each state of \mathcal{T} appears in only one of the *trees* of $\mathcal{ACD}(\mathcal{T})$.

► **Definition 4.12.** For each *proper tree* t_i of $\mathcal{ACD}(\mathcal{T})$ we say that t_i is *even* if $\nu_i(\varepsilon) \in \mathcal{F}$ and that it is *odd* if $\nu_i(\varepsilon) \notin \mathcal{F}$.

We say that the *alternating cycle decomposition* of \mathcal{T} is *even* if all the trees of maximal *height* of $\mathcal{ACD}(\mathcal{T})$ are *even*; that it is *odd* if all of them are *odd*, and that it is *ambiguous* if there are even and odd trees of maximal *height*.

► **Definition 4.13.** For each $\tau \in t_i$, $i = 1, \dots, r$, we define the *priority* of τ in t_i , written $p_i(\tau)$ as follows:

- If $\mathcal{ACD}(\mathcal{T})$ is *even* or *ambiguous*
 - If t_i is *even* ($\nu_i(\varepsilon) \in \mathcal{F}$), then $p_i(\tau) := \text{Depth}(\tau) = |\tau|$.
 - If t_i is *odd* ($\nu_i(\varepsilon) \notin \mathcal{F}$), then $p_i(\tau) := \text{Depth}(\tau) + 1 = |\tau| + 1$.
- If $\mathcal{ACD}(\mathcal{T})$ is *odd*
 - If t_i is *even* ($\nu_i(\varepsilon) \in \mathcal{F}$), then $p_i(\tau) := \text{Depth}(\tau) + 2 = |\tau| + 2$.
 - If t_i is *odd* ($\nu_i(\varepsilon) \notin \mathcal{F}$), then $p_i(\tau) := \text{Depth}(\tau) + 1 = |\tau| + 1$.

For $i = 0$, we define $p_0(\varepsilon) = 0$ if $\mathcal{ACD}(\mathcal{T})$ is *even* or *ambiguous* and $p_0(\varepsilon) = 1$ if $\mathcal{ACD}(\mathcal{T})$ is *odd*.

The assignation of priorities to nodes produces a labelling of the levels of each tree. It will be used to determine the priorities needed by a parity *transition system* to simulate T . The distinction between the cases $\mathcal{ACD}(\mathcal{T})$ even or odd is added only to obtain the minimal number of priorities in every case.

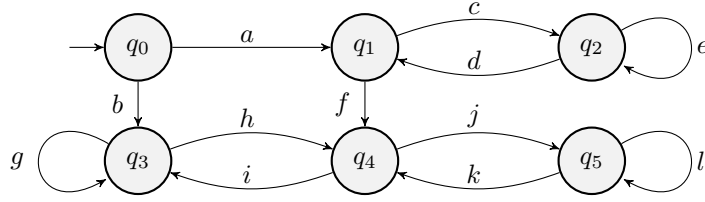
► **Example 4.14.** In figure 12 we represent a **transition system** $\mathcal{T} = (V, E, \text{Source}, \text{Target}, q_0, \mathcal{F})$ with $V = \{q_0, q_1, q_2, q_3, q_4, q_5\}$, $E = \{a, b, \dots, j, k\}$ and using the **Muller condition**

$$\mathcal{F} = \{\{c, d, e\}, \{e\}, \{g, h, i\}, \{l\}, \{h, i, j, k\}, \{j, k\}\}.$$

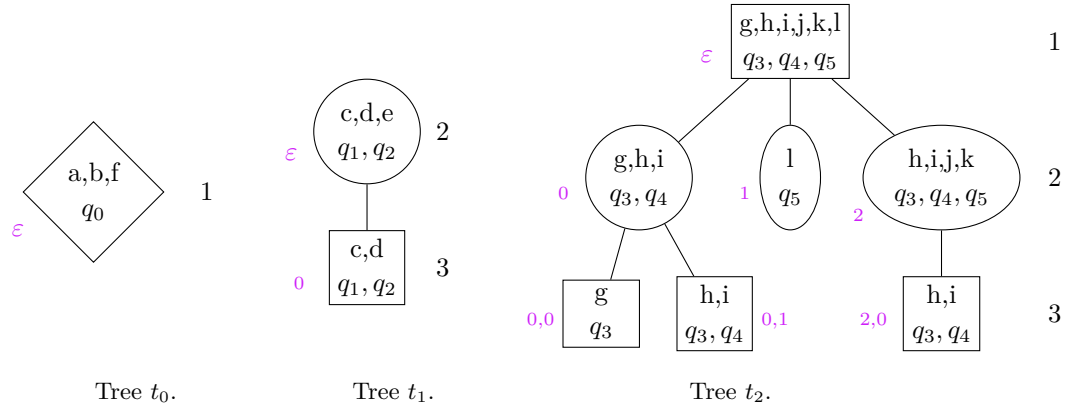
It has 2 strongly connected components (with vertices $S_1 = \{q_1, q_2\}$, $S_2 = \{q_3, q_4, q_5\}$), and a vertex q_0 that does not belong to any strongly connected component.

The **alternating cycle decomposition** of this transition system is shown in figure 13. It consists of two proper **trees**, t_1 and t_2 , corresponding to the strongly connected components of \mathcal{T} and the tree t_0 that corresponds to the edges not appearing in the strongly connected components.

We observe that $\mathcal{ACD}(\mathcal{T})$ is **odd** (t_2 is the highest tree, and it starts with a non-accepting **loop**). It is for this reason that we start labelling the levels of t_1 from 2 (if we had assigned priorities 0, 1 to the nodes of t_1 we would have used 4 priorities, when only 3 are strictly necessary).



■ **Figure 12** Transition system \mathcal{T} . $\mathcal{F} = \{\{c, d, e\}, \{e\}, \{g, h, i\}, \{l\}, \{h, i, j, k\}, \{j, k\}\}$.



■ **Figure 13** **Alternating cycle decomposition** of \mathcal{T} . The priority assigned to the nodes of each level of the trees is indicated at the right of the trees. Nodes with an even priority are drawn as circles and those with an odd priority as rectangles (excepting the special node forming the root of t_0). Each node τ is labelled with $\nu_i(\tau)$ and with $\text{States}_i(\tau)$. In **violet** the names of the nodes.

4.3 The alternating cycle decomposition transformation

We proceed to show how to use the **alternating cycle decomposition** of a **Muller transition system** to obtain a **parity transition system**. Let $\mathcal{T} = (V, E, \text{Source}, \text{Target}, I_0, \mathcal{F})$ be a **Muller transition system** and $(t_1, \nu_1), \dots, (t_r, \nu_r), (t_0, \nu_0)$ its **alternating cycle decomposition**.

► **Definition 4.15.** The *index* of an edge $e \in E$ (resp. of a state $q \in V$) in $\mathcal{ACD}(\mathcal{T})$ is the only number $j \in \{0, 1, \dots, r\}$ such that $e \in \nu_j(\varepsilon)$ (resp. $q \in \text{States}_j(\varepsilon)$).

► **Definition 4.16.** For an edge $e \in E$ of *index* j , for $i \in \{0, 1, \dots, r\}$ and a node $\tau \in t_i$ we define the *support* of e from τ as:

$$\text{Supp}(e, i, \tau) = \begin{cases} \text{The maximal ancestor } \sigma \sqsubseteq \tau \text{ such that } e \in \nu_i(\sigma), & \text{if } i = j. \\ \text{The root } \varepsilon \text{ of } t_j, & \text{if } i \neq j. \end{cases}$$

Intuitively, $\text{Supp}(e, i, \tau)$ is the highest node we visit if we want to go from τ to a node of the tree that contains e “in an optimal trajectory” (going up as little as possible). If we have to jump to another tree, we define $\text{Supp}(e, i, \tau)$ as the root of the destination tree.

► **Definition 4.17.** For $e \in E$ of *index* j , for $i \in \{0, 1, \dots, r\}$ and a node $\tau \in t_i$ we define the *seat* of e from τ as:

$\text{Seat}(e, i, \tau) =$ The maximal *descendant* σ of $\text{Supp}(e, i, \tau)$ for the relation \sqsubseteq such that $e \in \nu_j(\sigma)$.

For $q \in V$ of *index* k , the *seat* of q from τ reading e is:

$$\text{Seat}(q, e, i, \tau) = \begin{cases} \text{The maximal node } \sigma \in \text{Subtree}(\text{Seat}(e, i, \tau)) \text{ for the } \sqsubseteq \text{ relation} \\ \text{such that } q \in \text{States}_i(\sigma), & \text{if } j = k. \\ \text{The maximal node } \sigma \in t_k \text{ for } \sqsubseteq, & \text{if } j \neq k. \end{cases}$$

Intuitively, $\text{Seat}(e, i, \tau)$ is the node of t_j at which we arrive if we go from τ to a node that contains e , and then we descend as much as possible taking the leftmost node that contains e in each step waydown. The node $\text{Seat}(v, e, i, \tau)$ is where we arrive if we continue descending following the nodes where the vertex v appears (the leftmost branch if multiple options are available).

► **Definition 4.18** (From Muller to parity transition systems). Let $\mathcal{T} = (V, E, \text{Source}, \text{Target}, I_0, \mathcal{F})$ be a *Muller transition system* with *alternating cycle decomposition* $\mathcal{ACD}(\mathcal{T}) = \{(t_0, \nu_0), (t_1, \nu_1), \dots, (t_r, \nu_r)\}$. We define its *ACD-parity transition system* (or ACD-transformation) $\mathcal{P}_{\mathcal{ACD}}(\mathcal{T}) = (V_P, E_P, \text{Source}_P, \text{Target}_P, I'_0, p : E_P \rightarrow \mathbb{N})$ as follows:

- $V_P = \{(q, i, \tau) : i \in \{0, 1, \dots, r\}, \tau \in t_i, q \in \text{States}(\tau) \text{ and } \tau \text{ is maximal in } t_i \text{ (for } \sqsubseteq) \text{ among the nodes } \sigma \text{ such that } q \in \text{States}(\sigma)\}$.
- For each node $(q, i, \tau) \in V_P$ and each edge $e \in \text{Out}(q)$ we define an edge $e_{i, \tau}$ from (q, i, τ) . We set
 - $\text{Source}_P(e_{i, \tau}) = (q, i, \tau)$, where $q = \text{Source}(e)$.
 - $\text{Target}_P(e_{i, \tau}) = (q', k, \text{Seat}(q', e, i, \tau))$, where $q' = \text{Target}(e)$ and k is its *index*.
 - $p(e_{i, \tau}) = p_j(\text{Supp}(e, i, \tau))$, where j is the *index* of $\text{Supp}(e, i, \tau)$.
- $I'_0 = \{(q_0, i, \tau) : q_0 \in I_0, i \text{ the index of } q_0 \text{ and } \tau \text{ the maximal node in } t_i \text{ for } \sqsubseteq \text{ such that } q_0 \in \text{States}_i(\tau)\}$.

If \mathcal{T} is labelled by $l_V : V \rightarrow L_V$, $l_E : E \rightarrow L_E$, we label $\mathcal{P}_{\mathcal{ACD}}(\mathcal{T})$ by $l'_V((q, i, \tau)) = l_V(q)$ and $l'_E(e_{i, \tau}) = l_E(e)$.

The set of states of $\mathcal{P}_{\mathcal{ACD}}(\mathcal{T})$ is build as follows: for each state $q \in \mathcal{T}$ we consider the subtree of $\mathcal{ACD}(\mathcal{T})$ consisting of the nodes with q in its label, and we add a state for each branch of this subtree.

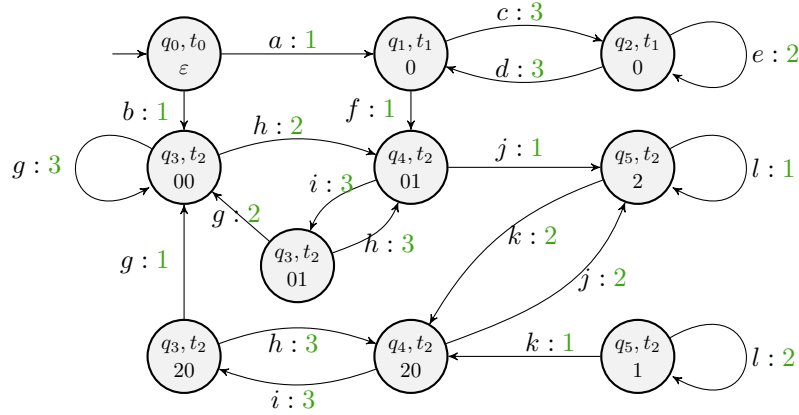
Intuitively, to define transitions in the transition system $\mathcal{P}_{\mathcal{ACD}}(\mathcal{T})$ we move simultaneously in \mathcal{T} and in $\mathcal{ACD}(\mathcal{T})$. We start from $q_0 \in I_0$ and from a node in $\mathcal{ACD}(\mathcal{T})$ that contains q_0

and none of its children does. When we take a transition e in \mathcal{T} , we go up the tree (or jump to another one) searching a node with e in its label, and we produce the priority corresponding to that level. Then, we go down again to $\text{Seat}(q', e, i, \tau)$, that is, to the leftmost node under our position containing e and q' .

► **Example 4.19.** In figure 14 we show the $\text{ACD-parity transition system } \mathcal{P}_{\text{ACD}(\mathcal{T})}$ of the transition system of example 4.14 (figure 12). States are labelled with the corresponding state q_j in \mathcal{T} , the tree t_i in $\text{ACD}(\mathcal{T})$ with i the index of q_j and a node $\tau \in t_i$ such that $q_j \in \text{States}_i(\tau)$ and τ is maximal for the prefix relation.

We have tagged the edges of $\mathcal{P}_{\text{ACD}(\mathcal{T})}$ with the names of edges of \mathcal{T} (even if it is not an automaton). These indicate the image of the edges by the morphism $\varphi : \mathcal{P}_{\text{ACD}(\mathcal{T})} \rightarrow \mathcal{T}$, and make clear the bijection between runs in \mathcal{T} and $\mathcal{P}_{\text{ACD}(\mathcal{T})}$.

We observe that every state q_j of \mathcal{T} gets multiplied by the number of maximal nodes (for \sqsubseteq) in a tree of $\text{ACD}(\mathcal{T})$ that contains q_j . Thus, in this example, q_0, q_1 and q_2 get multiplied by 1, q_3 by 3 and q_4 and q_5 by 2. The resulting parity transition system $\mathcal{P}_{\text{ACD}(\mathcal{T})}$ has therefore 10 states.

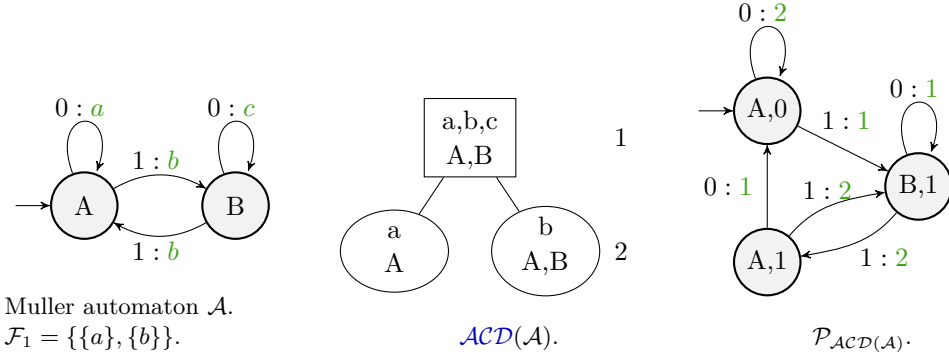


■ **Figure 14** Transition system $\mathcal{P}_{\text{ACD}(\mathcal{T})}$.

► **Example 4.20.** Let \mathcal{A} be the Muller automaton of example 2.4. Its alternating cycle decomposition has a single tree that coincides with the Zielonka tree of its Muller acceptance condition \mathcal{F}_1 (shown in figure 4). However, its $\text{ACD-parity transition system}$ has only 3 states, less than the product $\mathcal{A} \times \mathcal{Z}_{\mathcal{F}_1}$ (figure 11), as shown in figure 15.

► **Proposition 4.21** (Correctness). *Let $\mathcal{T} = (V, E, \text{Source}, \text{Target}, I_0, \mathcal{F})$ be a Muller transition system and $\mathcal{P}_{\text{ACD}(\mathcal{T})} = (V_P, E_P, \text{Source}_P, \text{Target}_P, I'_0, p : E_P \rightarrow \mathbb{N})$ its $\text{ACD-transition system}$. Then, there exists a locally bijective morphism $\varphi : \mathcal{P}_{\text{ACD}(\mathcal{T})} \rightarrow \mathcal{T}$. Moreover, if \mathcal{T} is a labelled transition system, then φ is a morphism of labelled transition systems.*

Proof. We define $\varphi_V : V_P \rightarrow V$ by $\varphi_V((q, i, \tau)) = q$ and $\varphi_E : E_P \rightarrow E$ by $\varphi_E(e_{i, \tau}) = e$. It is clear that this map preserves edges, initial states and labels. It is also clear that it is locally bijective, since we have define one initial state in $\mathcal{P}_{\text{ACD}(\mathcal{T})}$ for each initial state in \mathcal{T} , and by definition the edges in $\text{Out}((q, i, \tau))$ are in bijection with $\text{Out}(q)$. It induces therefore a bijection between the runs of the transition systems (fact 4.3).



■ **Figure 15** Muller automaton \mathcal{A} , its alternating cycle decomposition and its ACD-transformation.

Let us see that a run ϱ in \mathcal{T} is accepted if and only if $\varphi^{-1}(\varrho)$ is accepted in $\mathcal{P}_{\text{ACD}(\mathcal{T})}$. First, we remark that any infinite run ϱ of \mathcal{T} will eventually stay in a loop $l \in \text{Loop}(\mathcal{T})$ such that $\text{Inf}(\varrho) = l$, and therefore we will eventually only visit states corresponding to the tree t_i such that $l \subseteq \nu_i(\varepsilon)$ in the alternating cycle decomposition. We claim that there is a node $\tau \in t_i$ such that:

- $l \subseteq \nu_i(\tau)$.
- $l \not\subseteq \nu_i(\sigma)$ for every descendant σ , $\tau \sqsubseteq \sigma$.
- Eventually, we only visit states in $\text{States}_i(\tau)$, and we always eventually visit edges e such that $\text{Supp}(e, i, \sigma) = \tau$, for σ a descendant of τ .

The minimal priority produced by $\mathcal{P}_{\text{ACD}(\mathcal{T})}$ is therefore $p_i(\tau)$. Since there is no loop in $\nu_i(\tau)$ that includes l and alternates the membership of \mathcal{F} , we have that $l \in \mathcal{F}$ iff $\nu_i(\tau) \in \mathcal{F}$ iff $p_i(\tau)$ is even. We conclude that the run $\varphi^{-1}(\varrho)$ is accepted by $\mathcal{P}_{\text{ACD}(\mathcal{T})}$ if and only if \mathcal{T} accepts ϱ .

We only have to show that one such node $\tau \in t_i$ exists. The run ϱ is of the form $\varrho = \varrho_1 \varrho_2$, such that all edges appearing in ϱ_2 are in l . Let (q, i, α) be the state reached in $\mathcal{P}_{\text{ACD}(\mathcal{T})}$ after following the path $\varphi^{-1}(\varrho_1)$ and the first edge of $\varphi^{-1}(\varrho_2)$. Let τ_1 be the deepest node in t_i that is above α and contains l . Then, the node τ maximal for \sqsubseteq in $\text{Subtree}(\tau_1)$ containing l verifies the previous properties.

◀

From the remarks at the end of section 4.1, we obtain:

► **Corollary 4.22.** *If \mathcal{A} is a Muller automaton over Σ , the automaton $\mathcal{P}_{\text{ACD}(\mathcal{A})}$ is a parity automaton recognizing $\mathcal{L}(\mathcal{A})$. Moreover,*

- \mathcal{A} is deterministic if and only if $\mathcal{P}_{\text{ACD}(\mathcal{A})}$ is.
- \mathcal{A} is unambiguous if and only if $\mathcal{P}_{\text{ACD}(\mathcal{A})}$ is.
- \mathcal{A} is GFG if and only if $\mathcal{P}_{\text{ACD}(\mathcal{A})}$ is.

► **Corollary 4.23.** *If \mathcal{G} is a Muller game, then $\mathcal{P}_{\text{ACD}(\mathcal{G})}$ is a parity game that has the same winner than \mathcal{G} . The winning region of \mathcal{G} for a player $P \in \{\text{Eve}, \text{Adam}\}$ is $\mathcal{W}_P(\mathcal{G}) = \varphi(\mathcal{W}_P(\mathcal{P}_{\text{ACD}(\mathcal{G})}))$, being φ the morphism of the proof of proposition 4.21.*

4.4 Optimality of the alternating cycle decomposition transformation

In this section we prove the optimality of the [alternating cycle decomposition transformation](#), both for number of priorities (proposition 4.24) and for size (theorem 4.26). We use the same ideas as for proving the optimality of the [Zielonka tree automaton](#) in section 3.2.

► **Proposition 4.24** (Optimality of the number of priorities). *Let \mathcal{T} be a [Muller transition system](#) such that all its states are [accessible](#) and let $\mathcal{P}_{ACD(\mathcal{T})}$ be its [ACD-transition system](#). If \mathcal{P} is another [parity transition system](#) such that there is a [locally bijective morphism](#) $\varphi : \mathcal{P} \rightarrow \mathcal{T}$, then \mathcal{P} uses at least the same number of priorities than $\mathcal{P}_{ACD(\mathcal{T})}$.*

Proof. Let us suppose that $ACD(\mathcal{T})$ is [even](#) (similar proof if it is [odd](#)). Let t_i be a tree of maximal [height](#) h in $ACD(\mathcal{T})$, $b = \{\tau_1, \dots, \tau_h = \varepsilon\} \in \text{Branch}(t_i)$ a branch of t_i of maximal length and $l_j = \nu_i(\tau_j)$, $j = 1, \dots, h$. We fix $q \in \text{States}_i(\tau_1)$ and we write $\text{Loop}_{\mathcal{T}}(q) = \{w \in \text{Run}_{\mathcal{T}, q} \cap E^* : \text{First}(w) = \text{Last}(w) = q\}$, and for each $j = 1, \dots, h$ we choose $w_j \in \text{Loop}_{\mathcal{T}}(q)$ such that $\text{App}(w_j) = l_j$. We show as in the proof of proposition 3.9 that for every $v \in \text{Loop}_{\mathcal{T}}(q)$, the run $\varphi^{-1}((w_1 \dots w_h v)^\omega)$ must produce at least k different priorities in \mathcal{P} , having the smallest the same parity than the [depth](#) of τ_k . We conclude that \mathcal{P} uses at least h priorities, so at least as many as $\mathcal{P}_{ACD(\mathcal{T})}$.

In the case $ACD(\mathcal{T})$ is [ambiguous](#), $\mathcal{P}_{ACD(\mathcal{T})}$ uses $h + 1$ priorities. We can repeat the previous argument with two different maximal branches of respective maximal [even](#) and [odd](#) trees. We conclude that \mathcal{P} uses at least priorities in a range $[\mu, \mu + h] \cup [\eta, \eta + h]$, with μ even and η odd, so it uses at least $h + 1$ priorities. ◀

A similar proof, or an application of the results from [19] gives the following result:

► **Proposition 4.25.** *If \mathcal{A} is a deterministic automaton, the accessible part of $\mathcal{P}_{ACD(\mathcal{A})}$ uses the optimal number of priorities to recognize $\mathcal{L}(\mathcal{A})$.*

► **Theorem 4.26** (Optimality of the number of states). *Let \mathcal{T} be a (possibly [labelled](#)) [Muller transition system](#) such that all its states are [accessible](#) and let $\mathcal{P}_{ACD(\mathcal{T})}$ be its [ACD-transition system](#). If \mathcal{P} is another [parity transition system](#) such that there is a [locally bijective morphism](#) $\varphi : \mathcal{P} \rightarrow \mathcal{T}$, then $|\mathcal{P}_{ACD(\mathcal{T})}| \leq |\mathcal{P}|$.*

Proof of theorem 4.26

We follow the same steps as for proving theorem 3.13. We will suppose that all states of the [transition systems](#) considered are [accessible](#).

► **Definition 4.27.** *Let $\mathcal{T}_1, \mathcal{T}_2$ be [transition systems](#) such that there is a [morphism of transition systems](#) $\varphi : \mathcal{T}_1 \rightarrow \mathcal{T}_2$. Let $l \in \text{Loop}(\mathcal{T}_2)$ be a [loop](#) in \mathcal{T}_2 . An [l-SCC](#) of \mathcal{T}_1 (with respect to φ) is a non-empty [strongly connected subgraph](#) (V_l, E_l) of the subgraph $(\varphi_V^{-1}(\text{States}(l)), \varphi_E^{-1}(l))$ such that*

$$\begin{aligned} &\text{for every } q_1 \in V_l \text{ and every } e_2 \in \text{Out}(\varphi(q_1)) \cap l \\ &\text{there is an edge } e_1 \in \varphi^{-1}(e_2) \cap \text{Out}(q_1) \text{ such that } e_1 \in E_l. \end{aligned} \quad (\star)$$

That is, an [l-SCC](#) is a [strongly connected subgraph](#) of \mathcal{T}_1 in which all states and transitions correspond via φ to states and transitions appearing in the [loop](#) l . Moreover, given a [run](#) staying in l in \mathcal{T}_2 we can simulate it in the [l-SCC](#) of \mathcal{T}_1 (property (\star)).

► **Lemma 4.28.** *Let \mathcal{T}_1 and \mathcal{T}_2 be two [transition systems](#) such that there is a [locally surjective morphism](#) $\varphi : \mathcal{T}_1 \rightarrow \mathcal{T}_2$. Let $l \in \text{Loop}(\mathcal{T}_2)$ and $C_l = (V_l, E_l)$ be a non-empty [l-SCC](#) in \mathcal{T}_1 . Then, for every [loop](#) $l' \in \text{Loop}(\mathcal{T}_2)$ such that $l' \subseteq l$ there is a non-empty [l'-SCC](#) in C_l .*

Proof. Let $(V', E') = (V_l, E_l) \cap (\varphi_V^{-1}(\text{States}(l')), \varphi_E^{-1}(l'))$. We first prove that (V', E') is non-empty. Let $q_1 \in V_l \subseteq \text{States}(l)$. Let $\varrho \in \text{Run}_{T_2, \varphi(q)}$ be a finite run in \mathcal{T}_1 from $\varphi(q_1)$, visiting only edges in l and ending in $q_2 \in \text{States}(l')$. From the local surjectivity, we can obtain a run in $\varphi^{-1}(\varrho)$ that will stay in (V', E') and that will end in a state in $\varphi_V^{-1}(\text{States}(l'))$. The subgraph (V', E') clearly has property (\star) (for l').

We prove by induction on the size that any non-empty subgraph (V', E') verifying the property (\star) (for l') admits an l' -SCC. If $|V'| = 1$, then (V', E') forms by itself a **strongly connected graph**. If $|V'| > 1$ and (V', E') is not strongly connected, then there are vertices $q, q' \in V'$ such that there is no path from q to q' following edges in E' . We let

$$V'_q = \{p \in V' : \text{there is a path from } q \text{ to } p \text{ in } (V', E')\}; \quad E'_q = E' \cap \text{Out}(V'_q) \cap \text{In}(V'_q).$$

Since $q' \notin V'_q$, the size $|V'_q|$ is strictly smaller than $|V'|$. Also, the subgraph (V'_q, E'_q) is non-empty since $q \in V'_q$. The property (\star) holds from the definition of (V'_q, E'_q) . We conclude by induction hypothesis. ◀

► **Lemma 4.29.** *Let \mathcal{T} be a Muller transition system with acceptance condition \mathcal{F} and let \mathcal{P} be a parity transition system such that there is a locally bijective morphism $\varphi : \mathcal{P} \rightarrow \mathcal{T}$. Let t_i be a proper tree of $\mathcal{ACD}(\mathcal{T})$ and $\tau, \sigma_1, \sigma_2 \in t_i$ nodes in t_i such that σ_1, σ_2 are different children of τ , and let $l_1 = \nu_i(\sigma_1)$ and $l_2 = \nu_i(\sigma_2)$. If C_1 and C_2 are two l_1 -SCC and l_2 -SCC in \mathcal{P} , respectively, then $C_1 \cap C_2 = \emptyset$.*

Proof. Suppose there is a state $q \in C_1 \cap C_2$. Since $\varphi_V(q) \in \text{States}(l_1) \cap \text{States}(l_2)$, and l_1, l_2 are loops there are finite runs $\varrho_1, \varrho_2 \in \text{Run}_{\mathcal{T}, \varphi_V(q)}$ such that $\text{App}(\varrho_1) = l_1$, $\text{App}(\varrho_2) = l_2$. We can “simulate” these runs in C_1 and C_2 thanks to property (\star) , producing runs $\varphi^{-1}(\varrho_1)$ and $\varphi^{-1}(\varrho_2)$ in $\text{Run}_{\mathcal{P}, q}$ and arriving to $q_1 = \text{Last}(\varphi^{-1}(\varrho_1))$ and $q_2 = \text{Last}(\varphi^{-1}(\varrho_2))$. As C_1, C_2 are l_1, l_2 -SCC, there are finite runs $w_1 \in \text{Run}_{\mathcal{P}, q_1}$, $w_2 \in \text{Run}_{\mathcal{P}, q_2}$ such that $\text{Last}(w_1) = \text{Last}(w_2) = q$, so the runs $\varphi^{-1}(\varrho_1)w_1$ and $\varphi^{-1}(\varrho_2)w_2$ start and end in q . We remark that in \mathcal{T} the runs $\varphi(\varphi^{-1}(\varrho_1)w_1) = \varrho_1\varphi_E(w_1)$ and $\varphi(\varphi^{-1}(\varrho_2)w_2) = \varrho_2\varphi_E(w_2)$ start and end in $\varphi_V(q)$ and visit, respectively, all the edges in l_1 and l_2 . From the definition of $\mathcal{ACD}(\mathcal{T})$ we have that $l_1 \in \mathcal{F} \Leftrightarrow l_2 \in \mathcal{F} \Leftrightarrow l_1 \cup l_2 \notin \mathcal{F}$. As φ preserves the acceptance condition, the minimal priority produced by $\varphi^{-1}(\varrho_1)w_1$ has the same parity than that of $\varphi^{-1}(\varrho_2)w_2$, but concatenating both runs we must produce a minimal priority of the opposite parity, arriving to a contradiction. ◀

► **Definition 4.30.** *Let \mathcal{T} be a Muller transition system and $\mathcal{P}_{\mathcal{ACD}(\mathcal{T})} = (V_P, E_P, \text{Source}_P, \text{Target}_P, I'_0, \text{Acc}') its ACD-parity transition system. For each tree t_i of $\mathcal{ACD}(\mathcal{T})$, each node $\tau \in t_i$ and each state $q \in V$ of \mathcal{T} we define the numbers$*

$$\psi_{\tau, i, q} = |\{(q, i, \sigma) \in \mathcal{P}_{\mathcal{ACD}(\mathcal{T})} : \tau \sqsubseteq \sigma \text{ (}\sigma \text{ is a descendant of } \tau)\}|$$

$$\Psi_{\tau, i} = |\{(q, i, \sigma) \in \mathcal{P}_{\mathcal{ACD}(\mathcal{T})} : q \in V \text{ and } \tau \sqsubseteq \sigma \text{ (}\sigma \text{ is a descendant of } \tau)\}| = \sum_{q \in V} \psi_{\tau, i, q}.$$

We can understand these quantities as follows: for each q of \mathcal{T} we take the subtree $t_{i, q}$ of t_i consisting of those nodes with state q in their label. Then, $\psi_{\tau, i, q}$ is the number of branches under τ in the tree $t_{i, q}$. In particular $\psi_{\tau, i, q} = 0$ if $q \notin \text{States}_i(\tau)$. The value $\Psi_{\tau, i}$ is then the number of states in $\mathcal{P}_{\mathcal{ACD}(\mathcal{T})}$ associated to some node in t_i under τ .

► **Remark.** If we consider the root of the trees in $\mathcal{ACD}(\mathcal{T})$, then each $\Psi_{\varepsilon,i}$ is the number of states in $\mathcal{P}_{\mathcal{ACD}(\mathcal{T})}$ associated to this tree, i.e., $\Psi_{\varepsilon,i} = |\{(q, i, \sigma) \in \mathcal{P}_{\mathcal{ACD}(\mathcal{T})} : q \in V, \sigma \in t_i\}|$. Therefore

$$|\mathcal{P}_{\mathcal{ACD}(\mathcal{T})}| = \sum_{i=0}^r \Psi_{\varepsilon,i}.$$

Proof of theorem 4.26. Let $\mathcal{T} = (V, E, \text{Source}, \text{Target}, I_0, \mathcal{F})$ be a **Muller transition system**, $\mathcal{P}_{\mathcal{ACD}(\mathcal{T})}$ the **ACD-parity transition system** of \mathcal{T} and $\mathcal{P} = (V', E', \text{Source}', \text{Target}', I'_0, p' : E' \rightarrow \mathbb{N})$ a parity transition system such that there is a **locally bijective morphism** $\varphi : \mathcal{P} \rightarrow \mathcal{T}$.

First of all, we will prove that we can suppose that for each tree t_i of $\mathcal{ACD}(\mathcal{T})$ the three following conditions hold:

1. Each vertex of V belongs to a **strongly connected component**.
2. All leaves $\tau \in t_i$ verify $|\text{States}_i(\tau)| = 1$.
3. Nodes $\tau \in t_i$ verify $\text{States}_i(\tau) = \bigcup_{\sigma \in \text{Children}(\tau)} \text{States}_i(\sigma)$.

Indeed, let us define a transition system $\tilde{\mathcal{T}} = (V, \tilde{E}, \tilde{\text{Source}}, \tilde{\text{Target}}, \tilde{\mathcal{F}})$ by adding for each $q \in V$ two new edges, $e_{q,1}, e_{q,2}$ with $\tilde{\text{Source}}(e_{q,j}) = \tilde{\text{Target}}(e_{q,j}) = q$, for $j = 1, 2$. The modified **acceptance condition** $\tilde{\mathcal{F}}$ is given by: let $C \subseteq \tilde{E}$

- If $C \cap E \neq \emptyset$, then $C \in \tilde{\mathcal{F}} \Leftrightarrow C \cap E \in \mathcal{F}$ (the occurrence of edges $e_{q,j}$ does not change the **acceptance condition**).
- If $C \cap E = \emptyset$, if there are edges of the form $e_{q,1}$ in C , for some $q \in V$, then $C \in \tilde{\mathcal{F}}$. If all edges of C are of the form $e_{q,2}$, $C \notin \mathcal{F}$.

It is easy to verify that the **transition system** $\tilde{\mathcal{T}}$ verifies the previous conditions. We perform similar operations in \mathcal{P} , obtaining $\tilde{\mathcal{P}}$: we add a pair of edges $e_{q,1}, e_{q,2}$ for each vertex in \mathcal{P} , and we assign them priorities $\tilde{p}(e_{q,1}) = \eta + \epsilon$ and $\tilde{p}(e_{q,2}) = \eta + \epsilon + 1$, where η is the maximum of the priorities in \mathcal{P} and $\epsilon = 0$ if η is even, and $\epsilon = 1$ if η is odd. We can extend the **morphism** φ to $\tilde{\varphi} : \tilde{\mathcal{P}} \rightarrow \tilde{\mathcal{T}}$ conserving the **local bijectivity** by setting $\tilde{\varphi}_E(e_{q,j}) = e_{\varphi(q),j}$ for $j = 1, 2$. Finally, it is not difficult to verify that $\mathcal{P}_{\mathcal{ACD}(\tilde{\mathcal{T}})} = \tilde{\mathcal{P}}_{\mathcal{ACD}(\mathcal{T})}$, so in particular $|\mathcal{P}_{\mathcal{ACD}(\tilde{\mathcal{T}})}| = |\mathcal{P}_{\mathcal{ACD}(\mathcal{T})}|$. Consequently, $|\mathcal{P}_{\mathcal{ACD}(\tilde{\mathcal{T}})}| \leq |\tilde{\mathcal{P}}|$ implies $|\mathcal{P}_{\mathcal{ACD}(\mathcal{T})}| \leq |\mathcal{P}|$. If the transition systems are labelled, we can label the new edges $e_{q,j}$ with fresh letters a_j , so this transformation is correct for **labelled transition systems** too.

For now on, we take \mathcal{T} verifying the three conditions above. In particular, all trees are **proper trees** in $\mathcal{ACD}(\mathcal{T})$. It is also verified that for each $q \in V$ and $\tau \in t_i$ that is not a leaf, $\psi_{\tau,i,q} = \sum_{\sigma \in \text{Children}(\tau)} \psi_{\sigma,i,q}$. Therefore, for each $\tau \in t_i$ that is not a leaf

$$\Psi_{\tau,i} = \sum_{\sigma \in \text{Children}(\tau)} \Psi_{\sigma,i},$$

and for each leaf $\sigma \in t_i$ we have $\Psi_{\sigma,i} = 1$.

Vertices of V' are partitioned in the equivalence classes of the preimages by φ of the roots of the trees $\{t_1, \dots, t_r\}$ of $\mathcal{ACD}(\mathcal{T})$:

$$V' = \bigcup_{i=1}^r \varphi_V^{-1}(\text{States}_i(\varepsilon)) \quad \text{and} \quad \varphi_V^{-1}(\text{States}_i(\varepsilon)) \cap \varphi_V^{-1}(\text{States}_j(\varepsilon)) = \emptyset \text{ for } i \neq j.$$

► **Claim.** For each $i = 1, \dots, r$ and each $\tau \in t_i$, if C_τ is a non-empty $\nu_i(\tau)$ -SCC, then $|C_\tau| \geq \Psi_{\tau,i}$.

Let us suppose this claim holds. In particular $(\varphi_V^{-1}(\text{States}_i(\varepsilon)), \varphi_E^{-1}(\nu_i(\varepsilon)))$ verifies the property (\star) from definition 4.27, so from the proof of lemma 4.28 we deduce that it contains a $\nu_i(\varepsilon)$ -SCC and therefore $|\varphi_V^{-1}(\text{States}_i(\varepsilon))| \geq \Psi_{\varepsilon,i}$, so

$$|\mathcal{P}| = \sum_{i=1}^r |\varphi_V^{-1}(\text{States}_i(\varepsilon))| \geq \sum_{i=1}^r \Psi_{\varepsilon,i} = |\mathcal{P}_{\mathcal{ACD}(\mathcal{T})}|$$

concluding the proof.

► **Proof of the claim.** Let C_τ be a $\nu_i(\tau)$ -SCC, and let us prove $|C_\tau| \geq \Psi_{\tau,i}$ by induction on the **height of the node** τ . If τ is a leaf (in particular if its height is 1), $\Psi_{\tau,i} = 1$ and the claim is clear. If τ of height $h > 1$ is not a leaf, then it has children $\sigma_1, \dots, \sigma_k$, all of them of height $h - 1$. After lemmas 4.28 and 4.29, for $j = 1, \dots, k$ we can find disjoint $\nu_i(\sigma_j)$ -SCC included in C_τ , named C_1, \dots, C_k , so by induction hypothesis

$$|C_\tau| \geq \sum_{j=1}^k |C_j| \geq \sum_{j=1}^k \Psi_{\sigma_j,i} = \Psi_{\tau,i}.$$

◀

► **Remark.** From the hypothesis of theorem 4.26 we cannot deduce that there is a **morphism** from \mathcal{P} to $\mathcal{P}_{\mathcal{ACD}(\mathcal{T})}$ or vice-versa. To produce a counter-example it is enough to remember the “non-determinism” in the construction of $\mathcal{P}_{\mathcal{ACD}(\mathcal{T})}$. Two different orderings in the nodes of the trees of $\mathcal{ACD}(\mathcal{T})$ will produce two incomparable, but minimal in size parity transition systems that admit a **locally bijective morphism** to \mathcal{T} .

However, we can prove the following result:

► **Proposition 4.31.** *If $\varphi_1 : \mathcal{P}_{\mathcal{ACD}(\mathcal{T})} \rightarrow \mathcal{T}$ is the **locally bijective morphism** described in the proof of proposition 4.21, then for every state q in \mathcal{T} of **index** i :*

$$|\varphi_1^{-1}(q)| = \psi_{\varepsilon,i,q} \leq |\varphi^{-1}(q)|, \text{ for every locally bijective morphism } \varphi : \mathcal{P} \rightarrow \mathcal{T}.$$

Proof. It is enough to remark that if $q \in \text{States}_i(\tau)$, then any $\nu_i(\tau)$ -SCC C_τ of \mathcal{P} will contain some state in $\varphi^{-1}(q)$. We prove by induction as in the proof of the claim that $\psi_{\tau,i,q} \leq |C_\tau \cap \varphi^{-1}(q)|$. ◀

5 Applications

5.1 Determinisation of Büchi automata

In many applications, such as the synthesis of reactive systems for *LTL*-formulas, we need to have **deterministic** automata. For this reason, the determinisation of automata is usually a crucial step. Since McNaughton showed in [16] that Büchi automata can be transformed into Muller deterministic automata recognizing the same language, much effort has been put into finding an efficient way of performing this transformation. The first efficient solution was proposed by Safra in [21], producing a deterministic automaton using a **Rabin condition**. Due to the many advantages of **parity conditions** (simplicity, easy complementation of automata, they admit memoryless strategies for games, closeness under union and intersection...), determinisation constructions towards parity automata have been proposed too. In [20], Piterman provides a construction producing a parity automaton that in addition improves the state-complexity of Safra’s construction. In [22], Schewe differentiates two steps in Piterman’s

algorithm, providing two constructions from a non deterministic Büchi automaton \mathcal{B} : the first towards a [Rabin automaton](#) ($\mathcal{R}_{\mathcal{B}}$) and the second gives Piterman's parity automaton ($\mathcal{P}_{\mathcal{B}}$).

In this section we prove that there is a [locally bijective morphism](#) from $\mathcal{P}_{\mathcal{B}}$ to $\mathcal{R}_{\mathcal{B}}$, and therefore we would obtain a smaller parity automaton applying the [ACD-transformation](#) in the second step. We provide an example (example 5.7) in which the [ACD-transformation](#) provides a strictly better parity automaton.

From non-deterministic Büchi to deterministic Rabin automata

In [22], Schewe presents a construction of a deterministic [Rabin automaton](#) $\mathcal{R}_{\mathcal{B}}$ from a non-deterministic [Büchi automaton](#) \mathcal{B} . The set of states of the automaton $\mathcal{R}_{\mathcal{B}}$ is formed of what he calls [history trees](#). The number of history trees for a Büchi automaton of size n is given by the function $hist(n)$, that is shown to be in $o((1.65n)^n)$ in [22]. This construction is presented starting from a state-labelled Büchi automaton. A construction starting from a transition-labelled Büchi automaton can be found in [25]. In [6], Colcombet and Zdanowski proved the worst-case optimality of the construction.

► **Proposition 5.1** ([22]). *Given a non-deterministic [Büchi automaton](#) \mathcal{B} with n states, there is an effective construction of a deterministic [Rabin automaton](#) $\mathcal{R}_{\mathcal{B}}$ with $hist(n)$ states and using 2^{n-1} Rabin pairs that recognizes the language $\mathcal{L}(\mathcal{B})$.*

► **Proposition 5.2** ([6]). *For every $n \in \mathbb{N}$ there exists a non-deterministic [Büchi automaton](#) B_n of size n such that every deterministic Rabin automaton recognizing $\mathcal{L}(B_n)$ has at least $hist(n)$ states.*

From non-deterministic Büchi to deterministic parity automata

In order to build a deterministic [parity automaton](#) $\mathcal{P}_{\mathcal{B}}$ that recognizes the language of a given [Büchi automaton](#) \mathcal{B} , Schewe transforms the automaton $\mathcal{R}_{\mathcal{B}}$ into a parity one using what he calls a *later introduction record* (LIR). The LIR construction can be seen as adding an ordering (satisfying some restrictions) to the nodes of the [history trees](#). States of $\mathcal{P}_{\mathcal{B}}$ are therefore pairs of history trees with a LIR. In this way we obtain a similar parity automaton that with the Piterman's determinisation procedure [20]. The worst-case optimality of this construction was proved in [24, 25], generalising the methods of [6].

► **Proposition 5.3** ([22]). *Given a non-deterministic [Büchi automaton](#) \mathcal{B} with n states, there is an effective construction of a deterministic [parity automaton](#) $\mathcal{P}_{\mathcal{B}}$ with $O(n!(n-1)!)$ states and using $2n$ priorities that recognizes the language $\mathcal{L}(\mathcal{B})$.*

► **Proposition 5.4** ([24, 25]). *For every $n \in \mathbb{N}$ there exists a non-deterministic [Büchi automaton](#) B_n of size n such that $\mathcal{P}_{\mathcal{B}}$ has less than 1.5 times as many states as a minimal deterministic parity automaton recognizing $\mathcal{L}(B_n)$.*

A locally bijective morphism from $\mathcal{P}_{\mathcal{B}}$ to $\mathcal{R}_{\mathcal{B}}$

► **Proposition 5.5.** *Given a [Büchi automaton](#) \mathcal{B} and its determinisations to Rabin and parity automata $\mathcal{R}_{\mathcal{B}}$ and $\mathcal{P}_{\mathcal{B}}$, there is a [locally bijective morphism](#) $\varphi : \mathcal{P}_{\mathcal{B}} \rightarrow \mathcal{R}_{\mathcal{B}}$.*

Proof. Observing the construction of $\mathcal{R}_{\mathcal{B}}$ and $\mathcal{P}_{\mathcal{B}}$ in [22], we see that the states of $\mathcal{P}_{\mathcal{B}}$ are of the form (T, χ) with T an state of $\mathcal{R}_{\mathcal{B}}$ (a [history tree](#)), and $\chi : T \rightarrow \{1, \dots, |B|\}$ a LIR (that can be seen as an ordering of the nodes of T).

It is easy to verify that the mapping $\varphi_V((T, \chi)) = T$ defines a morphism $\varphi : \mathcal{R}_B \rightarrow \mathcal{P}_B$ (from fact 4.6 there is only one possible definition of φ_E). Since the automata are deterministic, φ is a [locally bijective morphism](#).

◀

► **Theorem 5.6.** *Let \mathcal{B} be a [Büchi automaton](#) and $\mathcal{R}_B, \mathcal{P}_B$ the deterministic Rabin and parity automata obtained by applying the Piterman-Schewe construction to \mathcal{B} . Then, the parity automaton $\mathcal{P}_{ACD(\mathcal{R}_B)}$ verifies $|\mathcal{P}_{ACD(\mathcal{R}_B)}| \leq |\mathcal{P}_B|$ and $\mathcal{P}_{ACD(\mathcal{R}_B)}$ uses a smaller number of priorities than \mathcal{P}_B .*

Proof. It is a direct consequence of propositions 5.5, 4.24 and theorem 4.26. ◀

► **Remark.** Furthermore, after proposition 4.25, $\mathcal{P}_{ACD(\mathcal{R}_B)}$ uses the optimal number of priorities to recognize $\mathcal{L}(\mathcal{B})$, and we directly obtain this information from the [alternating cycle decomposition](#) of $\mathcal{R}_B, ACD(\mathcal{R}_B)$.

In the [example 5.7](#) we show a case in which $|\mathcal{P}_{ACD(\mathcal{R}_B)}| < |\mathcal{P}_B|$ and for which the gain in the number of priorities is clear.

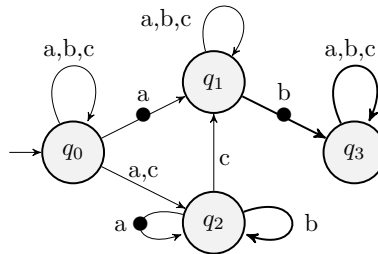
► **Remark.** In [6] and [25], the lower bounds for the determinisation of [Büchi automata](#) to Rabin and parity automata were shown using the family of [full Büchi automata](#), $\{\mathcal{B}_n\}_{n \in \mathbb{N}}$, $|\mathcal{B}_n| = n$. The automaton \mathcal{B}_n can simulate any other Büchi automaton of the same size. For these automata, the constructions $\mathcal{P}_{\mathcal{B}_n}$ and $\mathcal{P}_{ACD(\mathcal{R}_{\mathcal{B}_n})}$ coincide.

► **Example 5.7.** We present a non-deterministic Büchi automaton \mathcal{B} such that the [ACD-parity automaton](#) of \mathcal{R}_B has strictly less states and uses strictly less priorities than \mathcal{P}_B .

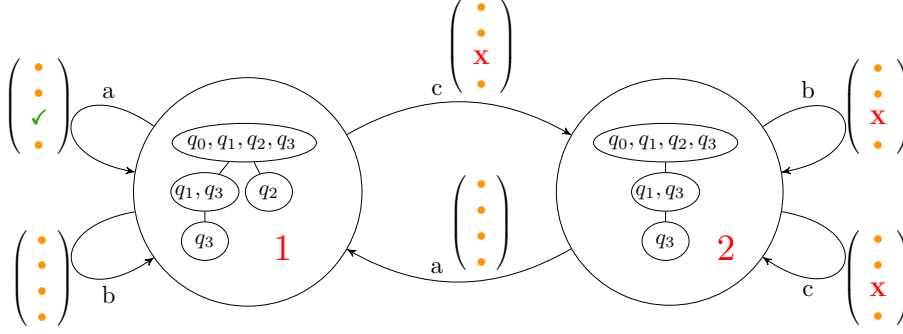
In figure 16 we show the automaton \mathcal{B} over the alphabet $\Sigma = \{a, b, c\}$. Accepting transitions for the [Büchi condition](#) are represented with a black dot on them. An accessible [strongly connected component](#) \mathcal{R}'_B of the determinisation to a [Rabin automaton](#) \mathcal{R}_B is shown in figure 17. It has 2 states that are [history trees](#) (as defined in [22]). There is a [Rabin pair](#) (E_τ, F_τ) for each node appearing in some [history tree](#) (four in total), and these are represented by an array with four positions. We assign to each transition and each position τ in the array the symbols \checkmark , \mathbf{X} , or \bullet depending on whether this transition belongs to E_τ , F_τ or neither of them, respectively (we can always suppose $E_\tau \cap F_\tau = \emptyset$).

In figure 18 there is the [alternating cycle decomposition](#) corresponding to \mathcal{R}'_B . We observe that the tree of $ACD(\mathcal{R}'_B)$ has a single branch of height 3. This is, the Rabin condition over \mathcal{R}'_B is already a [\[1, 3\]-parity condition](#) and $\mathcal{P}_{ACD(\mathcal{R}'_B)} = \mathcal{R}'_B$. In particular it has 2 states, and uses priorities in $[1, 3]$.

On the other hand, in figure 19 we show the automaton \mathcal{P}'_B , that has 3 states and uses priorities in $[3, 7]$. The whole automata \mathcal{R}_B and \mathcal{P}_B are too big to be pictured in these pages, but the three states shown in figure 19 are indeed accessible from the initial state of \mathcal{P}_B .

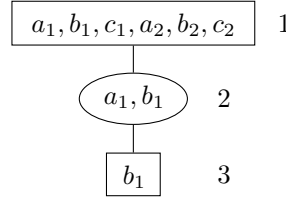


■ **Figure 16** Büchi automaton \mathcal{B} over the alphabet $\Sigma = \{a, b, c\}$.



■ **Figure 17** Component $\mathcal{R}'_{\mathcal{B}}$ of the automaton $\mathcal{R}_{\mathcal{B}}$. States are **history trees** and transitions are labelled with the input letter and an array representing the Rabin condition. There is one component in the arrays for each node appearing in some history tree, taking the order $\begin{pmatrix} \text{Root } \varepsilon \\ \text{Node 0} \\ \text{Node 1} \\ \text{Node 00} \end{pmatrix}$.

The components of the arrays can host a green checkmark \checkmark (if the node is in E_{τ}), a red cross \times (if the node is in F_{τ}) or an orange dot \bullet (if the node is not in E_{τ} or F_{τ}).



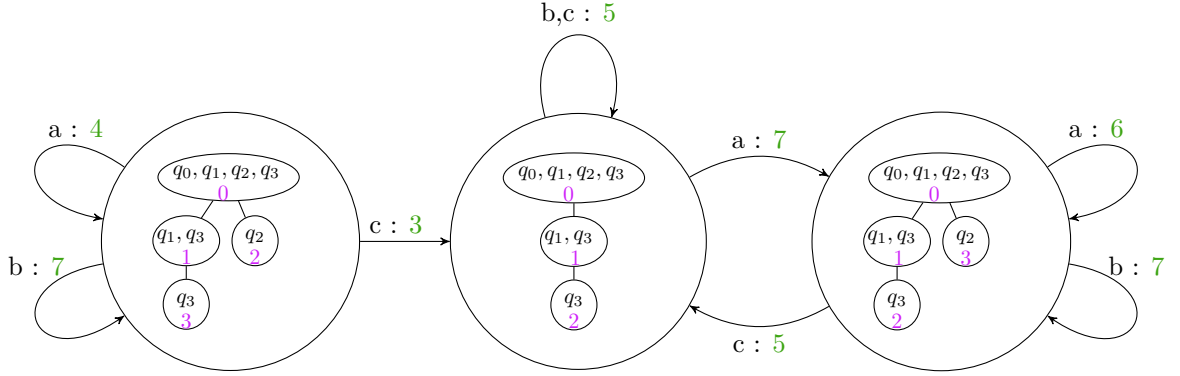
■ **Figure 18** The **alternating cycle decomposition** of $\mathcal{R}'_{\mathcal{B}}$, $\mathcal{ACD}(\mathcal{R}'_{\mathcal{B}})$. The labels a_1, b_1, c_1 correspond to transitions leaving the state 1 on figure 17 and a_2, b_2, c_2 those leaving state 2. We observe that this allows to substitute the **Rabin condition** on $\mathcal{R}'_{\mathcal{B}}$ for an equivalent $[1, 3]$ -parity condition on the same underlying automaton.

5.2 On relabelling of automata by acceptance conditions

In this section we use the information given by the **alternating cycle decomposition** to provide simple proofs of two results about the possibility to define different classes of acceptance conditions in an automaton. Theorem 5.12, first proven in [3], asserts that if we can define a Rabin and a Streett condition on top of an underlying automaton \mathcal{A} such that it recognizes the same language L with both conditions, then we can define a parity condition in \mathcal{A} recognizing L too. Theorem 5.13 states that if we can define Büchi and co-Büchi conditions on top of an automaton \mathcal{A} recognizing the language L , then we can define a **Weak** condition over \mathcal{A} such that it recognizes L .

► **Definition 5.8.** Given a Muller transition system \mathcal{T} , we say that its **alternating cycle decomposition** $\mathcal{ACD}(\mathcal{T})$ has

- **Rabin shape** if every node with even priority (**round nodes**) has at most one child.
- **Streett shape** if every node with odd priority (**square nodes**) has at most one child.
- **Parity shape** if every node has at most one child.
- **$[1, \eta]$ -parity shape** (resp. $[0, \eta - 1]$) if it has parity shape, every tree has **height** at most η and trees of height η are **odd** (resp. **even**).
- **Büchi shape** if it has $[0, 1]$ -parity shape and **co-Büchi shape** if it has $[1, 2]$ -parity shape.
- **Weak_k shape** if it has parity shape and every tree has **height** at most k .



■ **Figure 19** Component \mathcal{P}'_B of the automaton \mathcal{P}_B . States are ordered *history trees*, with the order labels in *violet*. It has three different states, since two different orders for the same *history tree* occur.

This terminology should be clear from the remarks of section 3.3. It follows from the definition that $\mathcal{ACD}(\mathcal{T})$ has *parity shape* if and only if it has *Rabin shape* and *Streett shape* simultaneously. We also remark that $\mathcal{ACD}(\mathcal{T})$ has *Weak_k shape* if and only if it has $[0, k]$ and $[1, k + 1]$ -*parity shape* (there are no trees of *height* $k + 1$ since they would be *even* and *odd* simultaneously).

► **Proposition 5.9.** *The alternating cycle decomposition $\mathcal{ACD}(\mathcal{T})$ of a transition system \mathcal{T} using a Muller condition \mathcal{F} has Rabin (resp. Streett / parity / Büchi / co-Büchi / Weak_k) shape, if and only if we can define a Rabin (resp. Streett / parity / Büchi / co-Büchi / Weak_k) condition over \mathcal{T} that is equivalent to \mathcal{F} over \mathcal{T} .*

Proof. We apply the construction explained in the proofs of propositions 3.14 and 3.15. ◀

► **Corollary 5.10.** *Given a transition system graph $\mathcal{T}_G = (V, E, \text{Source}, \text{Target}, I_0)$ and a Muller condition $\mathcal{F} \subseteq \mathcal{P}(E)$, we can define a parity condition $p : E \rightarrow \mathbb{N}$ equivalent to \mathcal{F} over \mathcal{T}_G if and only if we can define a Rabin condition R and a Streett condition S over E such that $(\mathcal{T}_G, \mathcal{F}) \simeq (\mathcal{T}_G, R) \simeq (\mathcal{T}_G, S)$*

Moreover, if the Rabin condition R uses r Rabin pairs and the Streett condition S uses s Streett pairs, we can take the parity condition p using priorities in

- $[1, 2r + 1]$ if $r \leq s$.
- $[0, 2s]$ if $s \leq r$.

Proof. For one implication, it suffices to remark that parity is a particular case of both Rabin and Streett conditions (a fact that becomes clear regarding the shapes of the $\mathcal{ACD}(\mathcal{T})$).

Conversely, if we can define Rabin and Streett conditions equivalent to \mathcal{F} over \mathcal{T} , then $\mathcal{ACD}(\mathcal{T})$ has both Rabin and Streett shape, and therefore *parity shape*. Moreover, trees of $\mathcal{ACD}(\mathcal{T})$ have *height* at most $\min\{2r + 1, 2s + 1\}$. The height $2r + 1$ can only be reached by *odd* trees, and $2s + 1$ only by *even* trees. ◀

► **Corollary 5.11.** *Given a transition system graph \mathcal{T}_G and a Muller condition \mathcal{F} over \mathcal{T}_G , there is an equivalent Weak_k condition over \mathcal{T}_G if and only if there are both $[0, k]$ and $[1, k + 1]$ -parity conditions equivalent to \mathcal{F} over \mathcal{T}_G .*

In particular, there is an equivalent *Weak condition* if and only if there are *Büchi* and *co-Büchi* conditions equivalent to \mathcal{F} over \mathcal{T}_G .

► **Remark.** It is important to notice that the previous results are stated for non-labelled transition systems. We must be careful when translating these results to automata and formal languages. For instance, in [3, Section 4] there is an example of a non-deterministic automaton \mathcal{A} , such that we can put on top of it Rabin and Streett conditions R and S such that $\mathcal{L}(\mathcal{A}, R) = \mathcal{L}(\mathcal{A}, S)$, but we cannot put a parity condition on top of it recognising the same language. However, we can find results analogous to those of propositions 5.9, 5.10 and 5.11 for deterministic automata.

► **Theorem 5.12** ([3, Theorem 7]). *Let \mathcal{A} be the [transition system graph](#) of a deterministic automaton with set of states Q . Let R be a Rabin condition over \mathcal{A} with r pairs and S a Streett condition over \mathcal{A} with s pairs such that $\mathcal{L}(\mathcal{A}, R) = \mathcal{L}(\mathcal{A}, S)$. Then, there exists a parity condition $p : Q \times \Sigma \rightarrow \mathbb{N}$ over \mathcal{A} such that $\mathcal{L}(\mathcal{A}, p) = \mathcal{L}(\mathcal{A}, R) = \mathcal{L}(\mathcal{A}, S)$. Moreover,*

- *if $r \leq s$, we can take p to be a $[1, 2r + 1]$ -parity condition.*
- *if $s \leq r$, we can take p to be a $[0, 2s]$ -parity condition.*

Proof. After corollary 5.10 we know that there is a parity condition p with the indicated priorities such that $(\mathcal{A}, p) \simeq (\mathcal{A}, R) \simeq (\mathcal{A}, S)$. This is, a run in (\mathcal{A}, p) is accepting if and only if it is also accepting in (\mathcal{A}, R) . Since in a deterministic automaton a word $u \in \Sigma^\omega$ determines a unique run $\mathcal{A}(u)$, this word will be accepted by (\mathcal{A}, p) if and only if it is also accepted by (\mathcal{A}, R) . ◀

► **Remark.** We have slightly improved the result in [3] by giving a finer description of the priorities used by the parity condition.

Using the same argument we obtain:

► **Theorem 5.13.** *Let \mathcal{A} be the [transition system graph](#) of a deterministic automaton and p and p' be $[0, k]$ and $[1, k + 1]$ -parity conditions respectively over \mathcal{A} such that $\mathcal{L}(\mathcal{A}, p) = \mathcal{L}(\mathcal{A}, p')$. Then, there exists a [Weak_k](#) condition W over \mathcal{A} such that $\mathcal{L}(\mathcal{A}, W) = \mathcal{L}(\mathcal{A}, p)$.*

In particular, there is a [Weak](#) condition W over \mathcal{A} such that $\mathcal{L}(\mathcal{A}, W) = L$ if and only if there are both Büchi and co-Büchi conditions B, B' over \mathcal{A} such that $\mathcal{L}(\mathcal{A}, B) = \mathcal{L}(\mathcal{A}, B') = L$.

6 Conclusions

We have presented a transformation that, given a Muller [transition system](#), provides an equivalent parity transition system that has minimal size and uses an optimal number of priorities among those which accept a [locally bijective morphism](#) to the original Muller transition system. In order to describe this transformation we have introduced the [alternating cycle decomposition](#), a data structure that arranges all the information about the acceptance condition of the transition system and the interplay between this condition and the structure of the system.

We have shown in section 5 how the alternating cycle decomposition can be useful to reason about acceptance conditions, and we hope that this representation of the information will be helpful in future works.

We have not discussed the complexity of effectively computing the [alternating cycle decomposition](#) of a Muller transition system. It is known that solving Muller games is PSPACE-complete when the acceptance condition is given as a list of accepting sets [12]. However, given a Muller game \mathcal{G} and $\mathcal{ACD}(\mathcal{G})$ (or just the [Zielonka tree](#) of the Muller condition), we have a transformation into a parity game of polynomial size on the size of \mathcal{G} , so solving Muller games with this extra information is in $\text{NP} \cap \text{co-NP}$. Consequently, unless PSPACE is contained in $\text{NP} \cap \text{co-NP}$, we cannot compute the [alternating cycle decomposition](#) of a Muller transition system in polynomial time.

References

- 1 J. Richard Büchi. On a decision method in restricted second order arithmetic. *Proc. Internat. Congr. on Logic, Methodology and Philosophy of Science*, 1960.
- 2 Udi Boker, Denis Kuperberg, Orna Kupferman, and Michał Skrzypczak. Nondeterminism in the presence of a diverse or unknown future. In *ICALP*, 2013.
- 3 Udi Boker, Orna Kupferman, and Avital Steinitz. Parityizing Rabin and Streett. In *FSTTCS*, 2010.
- 4 Olivier Carton and Max Michel. Unambiguous büchi automata. *Theoretical Computer Science*, 297(1):37 – 81, 2003.
- 5 Thomas Colcombet. The theory of stabilisation monoids and regular cost functions. In *ICALP*, 2009.
- 6 Thomas Colcombet and Konrad Zdanowski. A tight lower bound for determinization of transition labeled Büchi automata. In *ICALP*, 2009.
- 7 Jean-Michel Couvreur, Nasser Saheb, and Grégoire Sutre. An optimal automata approach to ltl model checking of probabilistic systems. In *LPAR*, 2003.
- 8 Stefan Dziembowski, Marcin Jurdziński, and Igor Walukiewicz. How much memory is needed to win infinite games? In *LICS*, 1997.
- 9 Yuri Gurevich and Leo Harrington. Trees, automata, and games. In *STOC*, 1982.
- 10 Thomas Henzinger and Nir Piterman. Solving games without determinization. In *CSL*, 2006.
- 11 Florian Horn. Random games. PhD Thesis, 2008.
- 12 Paul Hunter and Anuj Dawar. Complexity bounds for regular games. In *MFCS*, 2005.
- 13 Jan Křetínský, Tobias Meggendorfer, Clara Waldmann, and Maximilian Weininger. Index appearance record for transforming rabin automata into parity automata. In *TACAS*, 2017.
- 14 Denis Kuperberg and Michał Skrzypczak. On determinisation of good-for-games automata. In *ICALP*, 2015.
- 15 Christof Löding. Optimal bounds for transformations of omega-automata. In *FSTTCS*, 1999.
- 16 Robert McNaughton. Testing and generating infinite sequences by a finite automaton. *Information and control*, 9:521–530, 1966.
- 17 Andrzej W. Mostowski. Regular expressions for infinite trees and a standard form of automata. In *SCT*, 1984.
- 18 David E. Muller. Infinite sequences and finite machines. In *Symposium on Switching Circuit Theory and Logical Design*, 1963.
- 19 Damian Niwiński and Igor Walukiewicz. Relating hierarchies of word and tree automata. In *STACS*, 1998.
- 20 Nir Piterman. From nondeterministic buchi and streett automata to deterministic parity automata. In *LICS*, 2006.
- 21 Schmuel Safra. On the complexity of omega -automata. In *FOCS*, 1988.
- 22 Sven Schewe. Tighter bounds for the determinisation of büchi automata. In *Foundations of Software Science and Computational Structures*, pages 167–181, 2009.
- 23 Sven Schewe and Thomas Varghese. Tight bounds for the determinisation and complementation of generalised büchi automata. In *ATVA*, 2012.
- 24 Sven Schewe and Thomas Varghese. Determinising parity automata. In *MFCS*, 2014.
- 25 Thomas Varghese. Parity and generalised Büchi automata. Determinisation and complementation. PhD Thesis, 2014.
- 26 Klaus Wagner. On ω -regular sets. *Information and control*, 43(2):123–177, 1979.
- 27 Wiesław Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theoretical Computer Science*, 200(1-2):135–183, 1998.