

Chapter 8
Automata on infinite trees
Christof Löding

Contents

| | |
|---|-----|
| 1. Introduction | 265 |
| 2. Automata on infinite trees | 266 |
| 3. Constructions for complementation and simulation | 273 |
| 4. Decision problems | 285 |
| 5. Applications in logic | 288 |
| References | 297 |

1. Introduction

Infinite trees are a useful tool for modeling the possible executions of a discrete-state system. The branching in the tree either represents nondeterminism in the system, or different behaviours of an environment the system is interacting with. Starting from the root of the tree, a finite path represents a finite sequence of actions of the environment. The label of the tree node reached at the end of this sequence specifies the behaviour of the system. Taking this view, a set of infinite trees specifies a class of systems, and vice versa, a specification of a class of systems in terms of their allowed behaviours can be represented by a set of infinite trees.

Automata on infinite trees constitute a powerful and yet algorithmically feasible model for specifying sets of infinite trees. As an extension of automata on infinite words and automata on finite trees, this model was first investigated in [45] as a tool to solve decision problems in mathematical logic, namely for deciding the monadic second-order theory of two successor functions (the infinite binary tree). The seminal result of Rabin [45] states that a set of infinite labelled trees can be defined in monadic second-order logic if and only if it can be recognised by a finite automaton over infinite trees. Using a translation from formulas into automata gives a reduction from logical decision problems to questions on finite automata.

Based on this result, many further decidability results have been obtained, and a rich theory of automata on infinite trees, and its tight connection to games of infinite duration has been developed. There already exist various surveys and chapters of textbooks that cover many aspects of this theory. Most notably, the chapter on automata on infinite objects by Thomas [55], which gives an overview of the most fundamental results for automata on infinite words and infinite trees, and the later survey [56] that also includes

automata and logic on finite words and trees. The article of Zielonka [62] focuses on the theory of infinite games and its connection to automata on infinite trees. Some central aspects of automata on infinite trees are presented in the textbooks [26] and [43].

A distinguishing feature of the present article is the use of the model of alternating tree automata, which is not covered in the above mentioned presentations. Since alternating automata have become an important tool in the development of decision procedures in logic (see [58] for a comprehensive overview of many algorithms), the goal of this chapter is to give the key ideas for dealing with this model.

The chapter is structured as follows. In § 2 we introduce the basic terminology and the models of nondeterministic and alternating automata on infinite trees. For a compact representation we restrict ourselves to ordered trees with fixed finite branching degree. § 3 contains central constructions and proofs, namely the connection of tree automata and infinite games (§ 3.1), the complementation theorem for alternating automata (§ 3.2), and the simulation theorem for the transformation of alternating into nondeterministic automata (§ 3.3). § 3.4 provides some results on unambiguous and guidable automata, which are subclasses of nondeterministic automata. The complexity of decision problems for automata on infinite trees is considered in § 4, and applications in logic in § 5.

2. Automata on infinite trees

In this section we introduce the models of nondeterministic and alternating automata on infinite trees. We consider ordered trees of a fixed branching degree, meaning that all nodes have the same number of successors, and these successors are ordered. Automaton models for unordered trees without restrictions on the branching degree also exist but are beyond the scope of this chapter (see § 5.2 for references on this subject).

Let $k \in \mathbb{N}$ be a natural number and let $[k]$ denote the set $\{0, \dots, k-1\}$. We view the set $[k]^*$ of finite words over $[k]$ as the *domain* of an infinite k -ary tree. The *root* is the empty word ε , and for a node $u \in [k]^*$ and some $d \in [k]$ we call $u.d$ the d -successor of u . In the special case of $k = 2$ we speak of binary trees and sometimes call $u0$ the left successor of u , and $u1$ the right successor of¹ u .

A *path* π is an infinite sequence u_0, u_1, u_2, \dots of successive nodes starting in the root, i.e., $u_0 = \varepsilon$, and for each $j \geq 1$ there is some $d \in [k]$ with $u_{j+1} = u_j.d$.

Let A be an alphabet (a finite set of symbols). An infinite A -labelled k -ary *tree* is a mapping $t: [k]^* \rightarrow A$. In this chapter we only consider infinite trees, so we omit the word infinite when speaking about trees. Often A and/or k are clear from the context and we also omit A -labelled and/or k -ary.

We let $\mathcal{T}_{A,k}^\omega$ denote the set of all A -labelled k -ary trees. When we refer to binary trees, we usually omit the 2, and simply write \mathcal{T}_A^ω . In a tree t , a path π induces an infinite sequence of labels in A^ω . We let $t(\pi)$ denote this sequence, and sometimes

¹ We usually omit the dot in the concatenation of elements from $[k]$ if this does not raise any confusion.

identify a path with this infinite word. It should always be clear from the context to which meaning of a path we are referring to.

Besides k -ary trees, as introduced above, we also consider trees and forests in the graph-theoretic sense. We use the standard notion of forest: a directed graph (V, E) , where V is the set of vertices and E the set of edges, such that each node has at most one incoming edge and each node is reachable from a root (a node without incoming edge). A tree is a forest with only one root.

We now turn to the definition of automata for infinite trees. We start with nondeterministic automata, and then show how these are generalised by alternating automata.

2.1. Nondeterministic automata. A nondeterministic tree automaton on A -labelled k -ary trees consists of a transition structure and an acceptance condition. We separate the two things here because there are different acceptance conditions that are specified in different ways. The transition structure is of the form (Q, A, I, Δ) with a finite set Q of states, a set $I \subseteq Q$ of initial states, and a transition relation $\Delta \subseteq Q \times A \times Q^k$. Note that the branching degree of the trees is implicitly coded in the transition relation.

A run of a tree automaton on a tree $t \in \mathcal{T}_{A,k}^\omega$ is defined in terms of its transition structure: it is a tree $\rho \in \mathcal{T}_Q^\omega$ such that for each $u \in [k]^*$ we have $(\rho(u), t(u), \rho(u.0), \dots, \rho(u.k-1)) \in \Delta$. We say that ρ is *initial* if it starts in the initial state, i.e., if $\rho(\varepsilon) \in I$. If the run starts in some state q , then we say that ρ is a run from q .

As usual, a nondeterministic automaton accepts an input if there is some accepting run on it, where an accepting run has to be initial and it has to satisfy the acceptance condition. An abstract acceptance condition is given as a set $\text{Acc} \subseteq Q^\omega$ of infinite sequences of states. A run ρ is accepting if each path satisfies the acceptance condition, i.e., $\rho(\pi) \in \text{Acc}$ for each path π .

To remain within the scope of finite automata, concrete acceptance conditions have to be specified in a finite way. For automata on infinite trees, various acceptance conditions exist. Basically, we can take any acceptance condition that is used for automata on infinite words and use it for automata on infinite trees. In the following we list some of the main acceptance conditions and the way they are specified. For an infinite sequence π we let $\text{Inf}(\pi)$ denote the set of elements that have infinitely many occurrences in π .

- A *Büchi condition* [5] is given by a set $F \subseteq Q$ of accepting states. An infinite sequence π of states satisfies the Büchi condition if it contains infinitely many occurrences of states from F : $\text{Inf}(\pi) \cap F \neq \emptyset$.
- A *co-Büchi condition* is also given by a set $F \subseteq Q$ of accepting states. An infinite sequence π of states satisfies the co-Büchi condition if it contains finitely many occurrences of states from F : $\text{Inf}(\pi) \cap F = \emptyset$.
- A *Muller condition* [36] is given by a family $\mathcal{F} \subseteq 2^Q$ of sets of states. An infinite sequence π of states satisfies the Muller condition if it contains for some $F \in \mathcal{F}$ exactly the states from F infinitely often: $\text{Inf}(\pi) \in \mathcal{F}$.

- A *Rabin condition* [45] is given by a list $\langle (E_1, F_1), \dots, (E_k, F_k) \rangle$ of pairs of state sets. An infinite sequence π of states satisfies the Rabin condition if for some i it contains only finitely many occurrences of states from E_i and infinitely many occurrences of states from F_i : $\text{Inf}(\pi) \cap E_i = \emptyset$ and $\text{Inf}(\pi) \cap F_i \neq \emptyset$ for some $i \in \{1, \dots, k\}$.
- A *Streett condition* [54] is given by a list $\langle (E_1, F_1), \dots, (E_k, F_k) \rangle$ of pairs of state sets. An infinite sequence π of states satisfies the Streett condition if for all i it contains infinitely many occurrences of states from E_i or only finitely many occurrences of states from F_i : $\text{Inf}(\pi) \cap E_i = \emptyset \rightarrow \text{Inf}(\pi) \cap F_i = \emptyset$ for all $i \in \{1, \dots, k\}$.
- A *parity condition* [34] is given by a mapping $c: Q \rightarrow \mathbb{N}$ that assigns to each state a natural number, called *priority*. An infinite sequence π of states satisfies the parity condition if the maximal priority that appears infinitely often is even: $\max(\text{Inf}(c(\pi)))$ is even.

A nondeterministic tree automaton is composed of its transition structure and the acceptance component. For example, a parity tree automaton is of the form $\mathcal{A} = (Q, A, I, \Delta, c)$ and a Büchi tree automaton is of the form $\mathcal{A} = (Q, A, I, \Delta, F)$. Note that it is necessary to specify the type of the automaton because there are acceptance conditions which are specified in the same way but with different semantics.

A run ρ is *accepting* if the acceptance condition is satisfied on each path of the run. A tree t is accepted by \mathcal{A} if there is an accepting run of \mathcal{A} on t . The *language* accepted by \mathcal{A} is the set of all accepted trees: $T(\mathcal{A}) = \{t \in \mathcal{T}_{A,k}^\omega \mid t \text{ accepted by } \mathcal{A}\}$.

Example 2.1. Let $A = \{a, b\}$ and $T_{\exists b} \subseteq \mathcal{T}_{A,k}^\omega$ be the set of all binary trees containing at least one b . A Büchi tree automaton for this language uses two states $Q = \{q, q_b\}$, where q_b is used to guess a path leading to b , and q is used to label the remainder of the tree. The only initial state is q_b , and the transitions are (q_b, a, q_b, q) , (q_b, a, q, q_b) , (q_b, b, q, q) , (q, a, q, q) , and (q, b, q, q) . As long as the automaton reads a , it can continue the path labelled q_b to the left or to the right. If it finds some b on this path, it switches to q . To accept a tree this switch to q should finally happen, which is enforced by setting $F = \{q\}$.

Comparing the acceptance conditions, the Muller acceptance condition is the most general one of the conditions presented above. We call a tree language *regular* if it is accepted by some Muller tree automaton.

Relying on results for automata on infinite words, we can show that all the acceptance conditions that capture the class of regular ω -languages by means of *deterministic* automata on infinite words, lead to the same class of tree languages. Thus, the following theorem is a consequence of the corresponding results for deterministic automata on infinite words, and has already been observed in [45].

Theorem 2.1. *The classes of languages definable by nondeterministic Muller, Rabin, Streett, and parity tree automata are all the same.*

Proof. We illustrate the proof by showing how to transform a Muller tree automaton into a parity tree automaton. First of all, this is sufficient to prove the theorem because a parity condition is a special case of Muller, Rabin, and Streett conditions. Furthermore, the principle of the proof is generic, relying on results for word automata, such that it can easily be instantiated for other types of transformations.

Let $\mathcal{A} = (Q, A, I, \Delta, \mathcal{F})$ be a Muller automaton. Let $\text{Acc} \subseteq Q^\omega$ be the abstract acceptance condition, i.e., the set of all infinite state sequences satisfying the Muller condition \mathcal{F} . Since Q is a finite set, we can view Acc as an ω -language over the alphabet Q . It is easy to see that Acc is a regular ω -language. Hence, we can find a deterministic parity automaton $\mathcal{B} = (S, Q, \iota, \delta, c)$ over infinite words that accepts Acc (follows from [48], see also [58] or Chapter 6 of this work).

We now combine \mathcal{A} and \mathcal{B} using a product construction such that \mathcal{B} reads the state sequences along the paths of a run of \mathcal{A} . The acceptance condition is induced by \mathcal{B} . Since \mathcal{B} accepts a sequence of states of \mathcal{A} if and only if it satisfies the Muller condition, we end up with a parity tree automaton equivalent to \mathcal{A} .

Note that this kind of construction can also be used for other kinds of acceptance conditions. One only has to check that the abstract acceptance condition of the given automaton can be accepted by a deterministic word automaton of the target acceptance condition, and that this target acceptance condition can be transferred to the product automaton. \square

This result raises the question whether acceptance conditions that are less expressive for deterministic word automata, as for example the Büchi condition, are also less expressive for nondeterministic tree automata. Consider the language of all infinite words over $A = \{a, b\}$ that contain only finitely many occurrences of b . This language cannot be accepted by a deterministic Büchi word automaton. And indeed, it is not possible to construct a Büchi tree automaton for the language containing all trees in which every path contains only finitely many occurrences of b (see e.g., [56] for a proof).

Proposition 2.2 ([47]). *The language of all trees over $\{a, b\}$ that have only finitely many occurrences of b on each path, cannot be recognised by a Büchi tree automaton.*

The relation between recognisability of a word language and the induced tree language can be generalised as follows. For a language L of infinite words let $\text{Path}(L)$ be the language of all trees such that each path is labelled by a word in L . In [28] it is shown that a language of the form $\text{Path}(L)$ can be accepted by a nondeterministic Büchi tree automaton if and only if L can be accepted by a deterministic Büchi word automaton. In [41] this result is extended to parity automata:

Theorem 2.3 ([41]). *A tree language of the form $\text{Path}(L)$ can be accepted by a nondeterministic parity tree automaton using priorities i, \dots, j if and only if L can be accepted by a deterministic parity word automaton using priorities i, \dots, j .*

Intuitively, this result shows that nondeterminism in tree automata does not help if the tree language requires that a property is checked on all paths.

Before we introduce the more general model of alternating tree automata, we present some basic closure properties of the class of nondeterministic tree automata. The closure under union is easily shown by taking the disjoint union of two given Muller automata.

For the closure under intersection one can use a standard product construction, yielding an automaton over the Cartesian product of the two state spaces. The acceptance condition contains all sets of pairs of states such that the projections to the two components yield state sets that are accepting in the respective Muller automata. One should note here that constructions for the intersection are more involved for other types of acceptance conditions like Rabin or parity conditions, because the two given conditions cannot easily be rewritten as a condition of the same type on the product state space. However, one can rely on Theorem 2.1 to derive the closure properties for other acceptance conditions.

Remark 2.4. The class of languages definable by nondeterministic Muller tree automata is closed under union and intersection.

A projection is a mapping h from an alphabet A to an alphabet B . As usual, a projection is applied to a tree by applying it to each node, and to a set of trees by applying it to each tree from the set.

By applying the projection h to each label in the transitions of a nondeterministic automaton, we obtain the following remark.

Remark 2.5. For all of the acceptance conditions, the corresponding class of languages definable by nondeterministic tree automata is closed under projection.

The closure under complement is the most difficult problem for nondeterministic automata. A proof of this is given in § 3 based on constructions for alternating automata, which are defined below.

2.2. Alternating automata. In nondeterministic automata, each transition sends exactly one state to each successor node in the tree. Alternating automata relax this restriction: it is possible to send several states to the same successor, or to ignore some subtrees by not sending states to the corresponding node at all. To generalise nondeterministic automata we also allow nondeterministic choices. For example, it is possible to specify that the automaton sends states q_1 and q_2 to the first successor, or it sends q_3 to the first successor and q_4 and q_5 to the second successor. If several states are sent to the same successor, then these can be viewed as copies of the automaton that process the same subtree independently.

We use positive Boolean formulas to define such transitions, where an atom specifies which state is sent to which successor. For a set X let $\mathcal{B}^+(X)$ denote the set of positive Boolean formulas over X (composed of atoms from X and the operators \wedge, \vee). We let \mathbf{tt} denote the empty conjunction, which is always true, and we write \mathbf{ff} for the empty disjunction, which is always false. For $\varphi \in \mathcal{B}^+(X)$ and $Y \subseteq X$ we write $Y \models \varphi$ if φ evaluates to true with all elements from Y interpreted as true, and all elements of $X \setminus Y$ interpreted as false.

The transition structure of an alternating automaton is of the form (Q, A, I, δ) , where Q and A are as before, $I \in \mathcal{B}^+(Q)$ is an initial formula, and the transitions are defined by a function $\delta: Q \times A \rightarrow \mathcal{B}^+(Q \times [k])$. The acceptance conditions are specified in the same way as for nondeterministic automata.

Nondeterministic automata can be seen as a special case of alternating automata. We can write the transition relation Δ of a nondeterministic automaton as the function δ defined by

$$\delta(q, a) = \bigvee_{(q, a, q_0, \dots, q_{k-1}) \in \Delta} (q_0, 0) \wedge \dots \wedge (q_{k-1}, k-1).$$

For nondeterministic automata, we refer to the elements (q, a, q_1, \dots, q_k) of Δ as transitions. Similarly, we say that (q, a, P) with $P \subseteq Q \times [k]$ is a transition of an alternating automaton with transition function δ if $P \models \delta(q, a)$.

Before we formally describe runs, we give an example to illustrate the principle of alternating automata.

Example 2.2. Let $A = \{a, b\}$ and let T be the language consisting of all binary trees such that below each a there is some b . More precisely,

$$T = \{t \in \mathcal{T}_A^\omega \mid \forall u \in [2]^*(t(u) = a \implies \exists v \in [2]^*(t(uv) = b))\}.$$

We construct an alternating automaton that uses two states $Q = \{q, q_b\}$, where q is used to run over the whole tree, and whenever a node with label a is found, the automaton spawns a new copy in the state q_b that guesses a path to some b below the current node (the use of q_b is similar to Example 2.1). The initial formula corresponds to a single initial state $I = q$. The transition function is defined as follows:

$$\begin{aligned} \delta(q, a) &= (q, 0) \wedge (q, 1) \wedge ((q_b, 0) \vee (q_b, 1)); \\ \delta(q, b) &= (q, 0) \wedge (q, 1); \\ \delta(q_b, a) &= (q_b, 0) \vee (q_b, 1); \\ \delta(q_b, b) &= \mathbf{tt}. \end{aligned}$$

The state q always reproduces itself at both successors of a node by the formula $(q, 0) \wedge (q, 1)$. The state q_b only continues into one of the directions by the formula $(q_b, 0) \vee (q_b, 1)$. When q_b finds a node labelled b , then it stops using the formula \mathbf{tt} . Some examples for transitions of \mathcal{A} are $(q, a, \{(q, 0), (q, 1), (q_b, 0)\})$ and (q_b, b, \emptyset) .

As acceptance condition, we use a Büchi condition given by $F = \{q\}$. This means that all computations from a state q_b have to terminate at some point because otherwise we would obtain an infinite path that only finitely often visits F .

The notion of run for alternating automata is more complex because the automaton does not label each node of an input tree by exactly one state, as it is the case for nondeterministic automata. For this reason, a run R is a forest with vertices labelled by states and by nodes of the tree. The paths through the forest correspond to computations of the automaton along paths of the input tree.

Let \mathcal{A} be an alternating tree automaton with transition structure (Q, A, I, δ) , and let $t \in \mathcal{T}_{A,k}^\omega$ be a tree. A run of \mathcal{A} on t is a pair (R, ρ) of a forest $R = (V_R, E_R)$ and a

mapping $\rho: V_R \rightarrow Q \times [k]^*$ that associates to each node of V_R a state of \mathcal{A} and a node from the domain of t , such that the following conditions are satisfied.

- Each root of the forest is labelled by a pair (q, ε) , and the set $\{q_1, \dots, q_n\}$ of state labels of the roots of R satisfies the initial formula I .
- Let $v \in V_R$ be a node of R with $\rho(v) = (q, u)$, and let v_1, \dots, v_n be its successors. Then for each i , the label of v_i is of the form $\rho(v_i) = (q_i, u.d_i)$ with $d_i \in [k]$, and $(q, t(u), \{(q_i, d_i) \mid 1 \leq i \leq n\})$ is a transition of \mathcal{A} .

Example 2.3. Figure 2 shows the beginning of a possible run of the automaton from Example 2.2 on the tree from Figure 1. Note that the paths ending in $(q_b, 01)$ are finite paths of the run because (q_b, b, \emptyset) is a transition of the automaton.

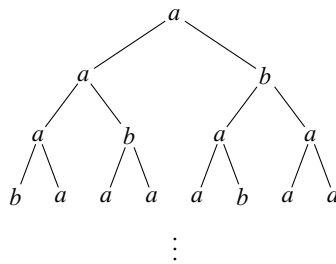


Figure 1. A labelled binary tree

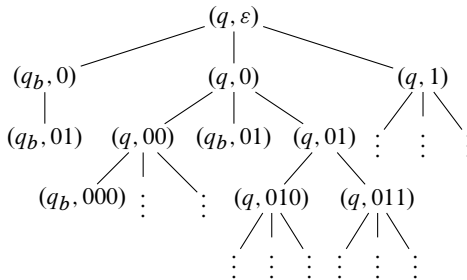


Figure 2. A run of the automaton from Example 2.2 on the tree from Figure 1

A run (R, ρ) is accepting if the state sequences that we obtain along the infinite paths through R , satisfy the acceptance condition. As before, we let $T(\mathcal{A})$ denote the set of all trees for which there is an accepting run of \mathcal{A} . This is also called the language accepted by \mathcal{A} .

Note that we have introduced the model using an initial formula. A simple construction shows that one can always work with a single initial state instead. We use this remark in some proofs to simplify constructions.

We also introduce a restricted version of alternating automata, so-called weak alternating automata [37]. Intuitively, in weak automata each state sequence can only switch a bounded number of times between acceptance and non-acceptance: there are no nested accepting and non-accepting cycles in the transition structure. This is formalised by using an ordered partition of the state space, i.e., Q is the disjoint union of sets Q_1, \dots, Q_m such that for each $q \in Q_i$ and each $a \in A$, all the states that occur in the formula $\delta(q, a)$ are in some Q_j for $i \leq j$. Furthermore, each Q_i is either classified as accepting or as rejecting. Note that the state labels on each path through a run of a weak automaton eventually remain in one of the Q_i . The path is accepting if this Q_i is an accepting component, otherwise it is rejecting.

The automaton presented in Example 2.2 is, in fact, a weak automaton with the partition $Q_1 = \{q\}$ and $Q_2 = \{q_b\}$, where Q_1 is accepting and Q_2 is rejecting.

Note that the acceptance condition of weak automata can be expressed as a Büchi condition using the set F consisting of the union of the accepting components, and as a co-Büchi condition using the set F consisting of the union of the rejecting components.

Weak alternating automata are interesting because they can be used to characterise a variant of monadic second-order logic. This relation is explained in more detail in § 5.1.

As for nondeterministic automata, we state some simple properties of alternating automata. The more involved constructions are presented in § 3. The possibility of using disjunctions and conjunctions in alternating automata, leads to simple constructions for the closure under union and intersection: take the disjoint union of the two given automata, and combine them by taking the disjunction or the conjunction of the initial formulas, respectively. Note that these constructions work for all the presented acceptance conditions (for Rabin and Streett conditions a simple adaption of the pairs is necessary, left as an exercise to the reader).

The operation of projection is more involved for alternating automata than for nondeterministic ones (see Remark 2.5). The standard way of dealing with projection is to transform an alternating automaton into a nondeterministic one. Such a transformation is the subject of § 3.3.

3. Constructions for complementation and simulation

In this section, we show how to complement alternating tree automata, and how to transform an alternating automaton into a nondeterministic one. This also settles the complementation problem for nondeterministic tree automata because we can complement a nondeterministic tree automaton as an alternating one, and then transform the result back into a nondeterministic automaton. This combined construction corresponds to the standard construction as presented in [56].

A major tool for proving the correctness of these constructions are infinite games. We introduce the required terminology and some basic results on games in § 3.1. In § 3.2 we show how to complement alternating automata, in § 3.3 we transform alternating automata into nondeterministic ones, and in § 3.4 we present some results on unambiguous and guidable automata.

3.1. Infinite games. The games we are interested in, are played on game graphs, also called arenas. An *arena* is a tuple $G = (V_E, V_A, E, c)$, where V_E is the set of vertices of player *Eva*, V_A is the set of vertices of player *Adam*, $E \subseteq V \times V$ for $V = V_E \cup V_A$ is the set of edges or moves, and $c: V \rightarrow C$ is a mapping that assigns to each vertex a colour from a finite set C of colours.

A *play* in G is an infinite sequence of vertices $\gamma \in V^\omega$ or a finite sequence of vertices that ends in a vertex without successors. One can imagine such a play being built up by the two players moving a token along the edges. Whenever the token is on a vertex of *Eva*, then she can choose the edge, otherwise *Adam* chooses.

Sometimes we are only interested in games starting in a specific initial vertex. In these cases we specify this vertex additionally to the other components of an arena.

Finite plays ending in vertices without successors are losing for the player who cannot move, that is, the player to whom the last vertex of the play belongs loses. The winner of infinite plays is specified by the winning condition: given a play γ , we look at the corresponding sequence $c(\gamma) \in C^\omega$ of colours. The winning condition specifies whether this sequence is winning for *Eva* or for *Adam*. Hence, formally a *winning condition* is given by the set $\text{Win} \subseteq C^\omega$ of infinite plays that are winning for *Eva*. A *game* is a pair $\mathcal{G} = (G, \text{Win})$ of an arena and a winning condition (for *Eva*) as above.

The central notion in the theory of infinite games is the one of winning strategy. Given a game, we are interested in the question if one of the players can guarantee to win, no matter how the opponent plays. This is captured by the notion of winning strategy. Formally, a *strategy* σ_E for *Eva* is a function $\sigma_E: V^*V_E \rightarrow V$ that takes a finite prefix of a play ending in a vertex of *Eva*, and outputs her next move, with the property that $\sigma_E(wv) = v'$ implies that $(v, v') \in E$. If a vertex of *Eva* does not have a successor, then the strategy is undefined. Similarly, a strategy for *Adam* is a function $\sigma_A: V^*V_A \rightarrow V$ such that $\sigma_A(wv) = v'$ implies that $(v, v') \in E$.

A (finite or infinite) play γ is said to be *played according to a strategy* σ_E for *Eva* if for each i with $\gamma(i) \in V_E$ the next vertex, if it exists, is obtained by applying the strategy: $\sigma_E(\gamma[0, i]) = \gamma(i + 1)$. For a strategy of *Adam* the definition is analogous.

Given a game $\mathcal{G} = (G, \text{Win})$ and an initial vertex v_0 , a strategy σ_E for *Eva* is a *winning strategy* from v_0 if all possible plays that start in v_0 and that are played according to σ_E are winning for *Eva* (and similarly for *Adam*).

The set of vertices from which *Eva* (*Adam*) has a winning strategy is called the *winning region* of *Eva* (of *Adam*). A *uniform winning strategy* for one of the players, is a strategy that is winning from each starting vertex within the winning region of the respective player.

A game is called *determined* if from each of the vertices one of the players has a winning strategy. A winning condition Win is called *determined* if every game with this winning condition is determined.

We are mainly interested in *parity games*. The parity condition is defined as for tree automata in § 2, that is, the colouring is of the form $c: V \rightarrow \mathbb{N}$. *Eva* wins a play γ if the maximal number that appears infinitely often in $c(\gamma)$ is even.

From this definition, it is clear that we do not have to explicitly specify the winning condition Win for parity games because it is already specified by the colouring function c . Hence, parity games are completely specified by the game arena (V_E, V_A, E, c) .

Parity games are important because the determinacy result already holds for a class of very simple strategies: if a player has a winning strategy from some node, then there is also a winning strategy that does not consider the history of the play for choosing the next move, but only the current vertex. Such strategies are called positional or memoryless.

Formally, a *positional strategy* for Eva is a function $\sigma_E: V_E \rightarrow V$ such that $(v, \sigma_E(v)) \in E$ for all $v \in V$. The same definition applies for Adam with V_A instead of V_E .

We call a game *positionally determined* if from each vertex one of the players has a positional winning strategy. The reason why parity games play a central role in the theory of automata and logics on infinite trees is the following theorem.

Theorem 3.1 ([17] and [35]). *Parity games are determined with uniform positional strategies for both players on their respective winning regions.*

There are various proofs of this result. The proofs in the survey articles [62] and [56] are based on an induction on the number of priorities. In [60] Walukiewicz gives a proof based on the concept of signatures or progress measures.

These different proofs of positional determinacy have led to different algorithms for computing the winning regions (and corresponding winning strategies) of parity games on finite graphs, with a time complexity in $\mathcal{O}(n^d)$ where n is the number of vertices and n the number of nodes of the parity game. Chapter 7 of [21] gives an overview of some of these approaches. The most obvious algorithm can directly be derived from the positional determinacy: on a finite graph one can guess a positional strategy and verify whether it is a winning strategy in polynomial time. By symmetry in the two players, we obtain that the problem of deciding whether a given vertex belongs to the winning region of Eva is in $\text{NP} \cap \text{co-NP}$. Currently, the best known algorithm for solving parity games runs in quasi-polynomial time [7]. In the same paper, it is also shown that parity games are fixed-parameter tractable, with the number of priorities being the parameter. A space efficient algorithm that also runs in quasi-polynomial time is presented in [25]. We summarise the above by the following theorem. For more precise statements on the two claims for complexities we refer to the papers [50], [7], and [25].

Theorem 3.2. *The problem of solving finite parity games is in $\text{NP} \cap \text{co-NP}$. There are deterministic algorithms solving parity games in time $\mathcal{O}(n^d)$ or in $n^{\mathcal{O}(\log(d))}$, where n is the number of vertices and d the number of priorities.*

The positional determinacy for parity games can be extended to Rabin games (with the definition of Rabin conditions as given in § 2) but here the positionality only holds for one player.

Theorem 3.3 ([27]). *Rabin games are determined and Eva has a positional winning strategy on her winning region.*

Since Rabin conditions and Streett conditions are complementary to each other, meaning that a Rabin condition for Eva corresponds to a Streett condition for Adam and vice versa, we obtain that positional strategies are sufficient for Adam in Streett games.

Corollary 3.4. *Streett games are determined and Adam has a positional winning strategy on his winning region.*

In Muller games both players might need memory. One can, however, show that finite memory only depending on the number of colours is sufficient for winning strategies, which then can be implemented by finite automata (see [21] and [62] for an overview of the main results).

Our next goal is to characterise the semantics of an alternating tree automaton \mathcal{A} in terms of a game, which has first been described in [22] for nondeterministic tree automata, and later been adapted for alternating tree automata [38]. In the original formulation, the players are called “automaton” and “pathfinder.” We stick to our standard terminology and call them Eva and Adam.

A first approximation of this game is as follows: given a tree t , the first player picks a run of \mathcal{A} on this tree, and then the second player chooses a path through this run. If the path satisfies the acceptance condition, then the first player wins, otherwise the second one wins. It is easy to see that the first player has a winning strategy if and only if the tree is accepted by \mathcal{A} . This game consists only of two rounds but the players choose very complex objects. We modify the game such that the players build the path incrementally: from a node on the path, which is of the form (q, x) with $x \in [k]^*$, Eva and Adam play the transition formula $\delta(q, t(x))$, where Eva resolves disjunctions and Adam resolves conjunctions. Resolving the formula results in some atom (q', d) with $d \in [k]$. The game continues in $(q', x.d)$.

For simplicity, we consider automata with a single initial state ι , which is sufficient as mentioned in § 2. The initial formula can be incorporated in the membership game but this makes the definition lengthy and less readable.

The formal definition of the *membership game* $\mathcal{G}_{\mathcal{A}, t}$ is as follows.

- The set of vertices is $V_{\mathcal{A}, t} = (Q \times [k]^*) \cup (\mathcal{B}^+(Q \times [k]) \times [k]^*)$.
- The vertices $(\psi, x) \in (\mathcal{B}^+(Q \times [k]) \times [k]^*)$ belong to Adam for conjunctions $\psi = \psi_1 \wedge \psi_2$ or if $\psi = \mathbf{tt}$, and to Eva for disjunctions and \mathbf{ff} . From the definition of the edge relation below, it follows that vertices from $Q \times [k]^*$ and (ψ, x) for an atom ψ only have one successor, and therefore it does not matter to which player they belong. The nodes with \mathbf{tt} and \mathbf{ff} have no successors, that is, Eva wins if a node with \mathbf{tt} is reached, and Adam wins if a node with \mathbf{ff} is reached.
- The initial vertex is (ι, ε) .

- To define the edges, let $q \in Q$, $x \in [k]^*$, $d \in [k]$, and $\psi \in \mathcal{B}^+(Q \times [k])$ be of the form $\psi = \psi_1 * \psi_2$ with $*$ $\in \{\wedge, \vee\}$. Then we have the following edges in $E_{\mathcal{A},t}$:

$$\begin{aligned} (q, x) &\longrightarrow (\delta(q, t(x)), x), \\ (\psi, x) &\longrightarrow (\psi_i, x) && \text{for } i \in \{1, 2\}, \\ ((q, d), x) &\longrightarrow (q, x.d). \end{aligned}$$

- The set of colours is the set Q of states of \mathcal{A} together with a colour \perp . To each node of the form (q, x) we assign the colour q , and to all other nodes the colour \perp . The winning condition is the adaption of the acceptance condition of \mathcal{A} : a play is winning for Eva if the state sequence that we obtain by removing \perp satisfies the acceptance condition.

This leads to a winning condition of the same type as the acceptance condition of \mathcal{A} . If \mathcal{A} uses a parity condition defined by the mapping c , then we can directly specify a parity condition for the membership game by assigning to each vertex (q, x) the priority $c(q)$, and to all other vertices the lowest priority in the image of c .

Figure 3 shows the reachable vertices of the initial part of the membership game for the automaton from Example 2.2 and the tree from Figure 1. The dashed ellipses indicate the nodes of the tree, which are ε , 0, and 1 in this example. Inside these dashed ellipses, the sub-game at the corresponding node is shown: a vertex (q, x) of the game is displayed as q inside the dashed ellipse for node x , and similarly for nodes of the form (ψ, x) . The vertices drawn with rounded corners belong to Eva, the others belong to Adam. For simplicity, conjunctions of three formulas are resolved in one step and not one conjunction symbol at the time, as in the formal definition.

Note that the label of the tree at a node influences the structure of the sub-game. Further, note that the game is tree shaped and acyclic, but it is not a tree. For example, there are two different ways of reaching the formula $(q_b, 0) \vee (q_b, 1)$ at the node 0. A strategy could make different decisions at this vertex depending on the way the play arrived there.

The following lemma shows in which sense the game $\mathcal{G}_{\mathcal{A},t}$ captures the semantics of tree automata.

Lemma 3.5 ([22] and [38]). *Eva has a winning strategy in $\mathcal{G}_{\mathcal{A},t}$ if and only if $t \in T(\mathcal{A})$.*

Proof. The main observation is that (q, a, P) is a transition of \mathcal{A} if and only if Eva can ensure, starting from a vertex $(\delta(q, a), x)$, to reach some vertex $((p, d), x)$ with $(p, d) \in P$. This can be shown by a simple induction on the structure of the formula $\delta(q, a)$.

Using this observation, one easily defines a winning strategy for Eva in $\mathcal{G}_{\mathcal{A},t}$ from an accepting run of \mathcal{A} on t and vice versa. \square

We have defined the game for alternating automata, such that at each tree node the two players play a local game for resolving the transition formulas. Consider the special case of nondeterministic automata. When viewing nondeterministic automata

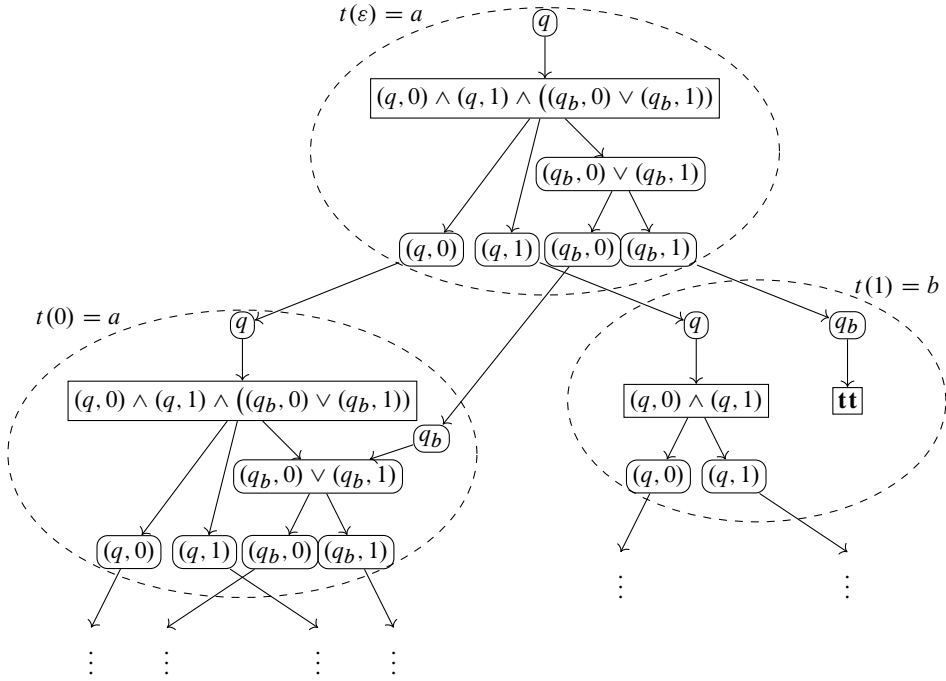


Figure 3. Initial reachable part of the membership game for the automaton from Example 2.2 and the tree from Figure 1

as alternating ones, then the transition formulas are disjunctions of conjunctions, where each conjunction contains one atom for each direction. In this case, the local game at each tree node can be viewed as Eva choosing a transition of the nondeterministic automaton (resolving the disjunction), and Adam choosing a direction (resolving the conjunction).

3.2. Complementation of alternating automata. Imagine that we transform the automaton \mathcal{A} into an automaton $\tilde{\mathcal{A}}$, such that in the membership game the winning condition is complemented, and the ownership of the vertices is swapped (nodes of Eva belong to Adam and vice versa). It is not difficult to see that in this modified game, Adam has a winning strategy if and only if Eva had a winning strategy in the original game. According to Lemma 3.5, this means that the automaton $\tilde{\mathcal{A}}$ accepts precisely those trees that have been rejected by \mathcal{A} .

The vertices in $\mathcal{G}_{\mathcal{A},t}$ that do not have a unique successor are precisely the vertices corresponding to non-atomic formulas from $\mathcal{B}^+(Q \times [k])$. Hence, if we exchange \wedge with \vee , and \mathbf{tt} with \mathbf{ff} in all the formulas of the automaton, then we basically dualise $\mathcal{G}_{\mathcal{A},t}$ as explained above. The formula obtained from φ by exchanging \wedge with \vee , and \mathbf{tt} with \mathbf{ff} is called the *dual formula* of φ , and is written as $\tilde{\varphi}$.

For an alternating tree automaton \mathcal{A} with transition function δ , we define the dual transition function $\tilde{\delta}$ by $\tilde{\delta}(q, a) = \delta(\widetilde{q}, a)$ for all states q and letters a . In this way, we can define the dual of an alternating transition structure (Q, A, I, δ) as $(Q, A, \tilde{I}, \tilde{\delta})$. The dual $\tilde{\mathcal{A}}$ of an alternating tree automaton \mathcal{A} is obtained by dualising its transition structure and its acceptance condition. An abstract acceptance condition Acc is dualised by simply taking its complement $Q^\omega \setminus \text{Acc}$. When working with concrete acceptance conditions, then dualisation sometimes requires to change the type of the automaton. For example, the dual of a Büchi condition specified by F , is described by the co-Büchi condition specified by F . The dual of a Rabin condition specified by $((E_1, F_1), \dots, (E_k, F_k))$, is captured by the same list of pairs interpreted as a Streett condition.

The other conditions that we have considered are self-dual. For a Muller condition specified by \mathcal{F} , the dual is specified by $2^Q \setminus \mathcal{F}$. For a parity condition specified by c , the dual is obtained by adding one to each priority (thus turning odd priorities even and vice versa). For weak automata, dualisation can be done by flipping the status of the components between accepting and rejecting.

Since dualisation basically amounts to flipping the roles of the two players in the membership game, we can conclude the following.

Lemma 3.6. *Eva has a winning strategy in $\mathcal{G}_{\mathcal{A},t}$ if and only if Adam has a winning strategy in $\mathcal{G}_{\tilde{\mathcal{A}},t}$.*

Combining Lemma 3.6 and Lemma 3.5, we obtain the following complementation theorem for alternating tree automata.

Theorem 3.7 ([38]). *Let \mathcal{A} be an alternating tree automaton and $\tilde{\mathcal{A}}$ be its dual. Then $\mathcal{T}_{\mathcal{A},k}^\omega \setminus T(\mathcal{A}) = T(\tilde{\mathcal{A}})$.*

3.3. Simulation of alternation by nondeterminism. Before we turn to the case of alternating parity tree automata, we state a simpler result for alternating Büchi automata due to Miyano and Hayashi.² The idea is to use an extension of the subset construction, often called *breakpoint construction* (see also Chapter 6 of this work).

Theorem 3.8 ([32]). *For each alternating Büchi tree automaton with n states one can construct an equivalent nondeterministic Büchi tree automaton with at most 3^n states.*

Proof idea. Let $\mathcal{A} = (Q, A, \iota, \delta, F)$ be an alternating Büchi tree automaton. We construct a nondeterministic Büchi tree automaton $\mathcal{A}' = (Q', A, \iota', \Delta', F')$ as follows: Q' consists of pairs $\langle P, M \rangle$, where $M \subseteq P \subseteq Q$. One can see this as a set of states P , in which some states are marked (those in M). Intuitively, the marked states correspond to paths that are still waiting for an accepting state. The transition relation is defined on the first component by a standard subset construction that removes the conjunctive branching from the alternating automaton. On the second component, a similar construction is performed, but in each step the accepting states are removed

² Miyano and Hayashi gave the construction for alternating word automata but the extension to trees is straightforward.

from the set. When the second component is empty, then all computations along the path that are collected in P have seen an accepting state. In the next step, all states are marked again, and the procedure restarts. A state sequence is accepting if the second component becomes empty infinitely often. \square

We now turn to the full simulation theorem. The proof strategy that we follow here, corresponds to the one for complementing nondeterministic parity tree automata as presented in [22] and [56]. It relies on positional determinacy of parity games, and determinisation of automata on infinite words.

As in the definition of the membership game, we work with automata having a single initial state. For the remainder of this section we fix an alternating parity tree automaton $\mathcal{A} = (Q, A, \iota, \delta, c)$.

Our goal is to construct a nondeterministic tree automaton that can check for a tree t , whether there is some successful run of \mathcal{A} on t . The difficulty is that such a run can have many different computations along a single path of the tree, while the nondeterministic automaton has exactly one computation along each path. In §3.1 we have seen that there is a correspondence between runs and strategies for Eva in the membership game. Thus, in the following we prefer to work with strategies instead of runs. Using the positional determinacy of parity games (Theorem 3.1), we can restrict our attention to rather simple objects: positional strategies.

A nondeterministic automaton can guess such a positional strategy, and at the same time verify that it is a winning strategy for Eva. We show this in two steps. First, we consider trees that are annotated by additional information coding a positional strategy for Eva. We show that a deterministic automaton can verify whether the annotation of t codes a winning strategy for Eva in $\mathcal{G}_{\mathcal{A},t}$. The nondeterministic automaton that guesses the strategy is then obtained by projection.

A positional strategy for Eva in $\mathcal{G}_{\mathcal{A},t}$ contains the following information: for each position of the form (ψ, x) where $\psi \in \mathcal{B}^+(Q \times [k])$ is a disjunction $\psi = \psi_1 \vee \psi_2$ and $x \in [k]^*$, the strategy has to choose a successor (ψ_1, x) or (ψ_2, x) . Hence, a positional strategy σ for Eva can be defined by specifying for each $x \in [k]^*$ a partial function $\hat{\sigma}_x: \mathcal{B}^+(Q \times [k]) \rightarrow \mathcal{B}^+(Q \times [k])$ such that for each $\psi = \psi_1 \vee \psi_2$ either $\hat{\sigma}_x(\psi) = \psi_1$ or $\hat{\sigma}_x(\psi) = \psi_2$. The function $\hat{\sigma}_x$ is partial because it only specifies values for disjunctive formulas. We call these partial functions $\hat{\sigma}_x$ local strategies, and we let Σ denote the set of all local strategies. Note that Σ is finite.

We can represent a positional strategy σ for Eva in $\mathcal{G}_{\mathcal{A},t}$ by annotating each node of t with the local strategy $\hat{\sigma}_x$. Looking at Figure 3, we basically annotate each node of t by the subgame at this node with one outgoing edge selected for each vertex of Eva.

Formally, a tree representation of a positional strategy for Eva is a tree $s \in \mathcal{T}_{A \times \Sigma, k}^\omega$. Such a tree s represents a strategy in the game $\mathcal{G}_{\mathcal{A},t}$ for the tree t , obtained by projecting s to the A -component of the labels. The central lemma for the simulation construction states that we can construct a parity tree automaton accepting precisely the representations of winning strategies of Eva.

Lemma 3.9. *There is a parity tree automaton \mathcal{A}' accepting the tree representations of positional winning strategies for Eva. The number of states of \mathcal{A}' is in $2^{\mathcal{O}(nd \log(nd))}$, and the number of priorities in $\mathcal{O}(nd)$, where n is the number of states of \mathcal{A} , and d is the number of priorities of \mathcal{A} .*

Proof. Let s be a tree representation of a positional strategy for Eva. Let t be the corresponding tree in $\mathcal{T}_{A,k}^\omega$ obtained by projection to the A -component, and let σ be the strategy of Eva in $\mathcal{G}_{A,t}$ represented by s .

The parity tree automaton that we construct, has to verify that all plays played according to σ are winning for Eva. We first need an auxiliary definition allowing us to talk about the plays that are possible according to σ .

Consider Figure 3, and assume that a local strategy at node ε chooses the edge to $(q_b, 0)$ from the disjunctive formula. The set of atoms that can be reached in the local game at ε according to this local strategy is $\{(q, 0), (q, 1), (q_b, 0)\}$. In general, we define the set $\text{atoms}(\hat{\sigma}, \psi)$ of atoms that a local strategy $\hat{\sigma}$ induces, starting from some element $\psi \in \mathcal{B}^+(Q \times [k])$:

$$\text{atoms}(\hat{\sigma}, \psi) = \begin{cases} \{(q, i)\} & \text{if } \psi = (q, i) \text{ is an atom,} \\ \text{atoms}(\hat{\sigma}, \psi_1) \cup \text{atoms}(\hat{\sigma}, \psi_2) & \text{if } \psi = \psi_1 \wedge \psi_2, \\ \text{atoms}(\hat{\sigma}, \hat{\sigma}(\psi)) & \text{if } \psi = \psi_1 \vee \psi_2. \end{cases}$$

Given a state q and a label $a \in A$, we let $\hat{\sigma}(q, a) = \text{atoms}(\hat{\sigma}, \delta(q, a))$ be the set of possible atoms that can be reached by $\hat{\sigma}$ from q for input letter a .

Now, we reduce the problem of checking all plays that are played according to σ , to a problem over infinite words: each path in s corresponds to an infinite sequence over $(A \times \Sigma \times [k])$, where the last $[k]$ -component indicates the direction the path takes in each step. We say that such a sequence $(a_0, \hat{\sigma}_0, i_0)(a_1, \hat{\sigma}_1, i_1) \cdots \in (A \times \Sigma \times [k])^\omega$ is \mathcal{A} -accepting if each state sequence $q_0 q_1 \cdots$ that starts in the initial state ($q_0 = \iota$), and is consistent with the strategy ($(q_{j+1}, i_j) \in \hat{\sigma}_j(q_j, a)$) satisfies the acceptance condition of \mathcal{A} . These conditions state that $q_0 q_1 \cdots$ corresponds to the state sequence of a play according to σ along the path $i_0 i_1 \cdots$. We conclude that σ is a winning strategy if and only if all paths through s correspond to \mathcal{A} -accepting sequences.

The set L of all \mathcal{A} -accepting sequences over $(A \times \Sigma \times [k])$ is a regular language of infinite words: a nondeterministic Büchi automaton can guess a sequence $q_0 q_1 \cdots$ that starts in the initial state, is consistent with the strategy, but does not satisfy the acceptance condition of \mathcal{A} . We obtain a deterministic parity word automaton \mathcal{D} for L by first determinising this Büchi automaton, and then complementing the resulting deterministic parity automaton (incrementing all priorities by 1). From this word automaton, one easily constructs a parity tree automaton $\mathcal{A}' = (Q', A \times \Sigma, \iota', \Delta', c')$ accepting all $(A \times \Sigma)$ -trees in which all paths correspond to \mathcal{A} -accepting sequences. The automaton \mathcal{A}' simulates \mathcal{D} along all paths: $(q, (a, \hat{\sigma}), q_1, \dots, q_k)$ is a transition of \mathcal{A}' if \mathcal{D} maps q to q_i for the input letter $(a, \hat{\sigma}, i)$ for each $i \in [k]$.

The size of \mathcal{A}' is determined by the size of \mathcal{D} , where \mathcal{D} was obtained from a non-deterministic Büchi automaton by determinization and complementation. The complementation step for deterministic parity automaton is only a matter of adapting the priority mapping. Hence, the size of \mathcal{D} is determined by the size of the Büchi automaton and the complexity of the determinisation construction. The Büchi automaton guesses a sequence $q_0q_1 \cdots$ using the states of \mathcal{A} . The local conditions are verified by the transitions, i.e., a transition $q \xrightarrow{(a, \hat{\sigma}, i)} q'$ is only possible if $(q', i) \in \hat{\sigma}(q, a)$. To check that $q_0q_1 \cdots$ does not satisfy the acceptance condition of \mathcal{A} , the automaton guesses a position and an odd priority m , and verifies that from this position onwards m is the biggest priority that occurs, and that m occurs infinitely often. Overall, this can be implemented using $n \cdot d$ states, where d is the number of priorities of \mathcal{A} (see also Remark 2.9 of [58]). Determinisation yields a deterministic parity word automaton with at most $2^{\mathcal{O}(nd \log(nd))}$ states and $\mathcal{O}(nd)$ priorities [48]. \square

The simulation result is a direct consequence of Lemma 3.9.

Theorem 3.10 ([18] and [39]). *For each alternating parity tree automaton, one can construct a nondeterministic parity tree automaton accepting the same language. The number of states of the nondeterministic automaton is bounded by $2^{\mathcal{O}(nd \log(nd))}$, where n is the number of states, and d is the number of priorities of the given alternating automaton.*

Proof. Given \mathcal{A} , we apply Lemma 3.9 and obtain a parity tree automaton \mathcal{A}' accepting the tree representations of winning strategies of Eva. Omitting the Σ -component from the labels in the transitions, yields a (nondeterministic) parity tree automaton that accepts all trees $t \in \mathcal{T}_{\mathcal{A},k}^\omega$ on which Eva has a winning strategy in $\mathcal{G}_{\mathcal{A},t}$. According to Lemma 3.5, this is exactly the language of \mathcal{A} . \square

By combining Theorem 3.10 and Theorem 3.7, we obtain a complementation procedure for nondeterministic tree automata.

Corollary 3.11 ([45]). *For each nondeterministic parity tree automaton, one can construct a nondeterministic parity tree automaton accepting the complement language.*

The simulation theorem and the closure of regular languages of infinite trees under complement is a strong result. We have seen that the proof is based on the determinisation theorem for automata on infinite words, and on the positional determinacy of parity games. Since the proof is constructive, we can also use it in connection with algorithmic questions for parity tree automata. This subject is considered in § 4.

The complementation result was already shown by Rabin in [45] also relying on McNaughton’s theorem on determinisation of ω -automata (as used in Lemma 3.9) but not on positional determinacy for membership games.

The simulation theorem for transforming alternating automata into nondeterministic ones, was first proven by Emerson and Jutla in [18] for alternating Streett tree automata, using the techniques presented in this chapter. In [39] Muller and Schupp give a proof that only assumes determinacy of membership games, but does not make

any assumptions on the shape of the strategies (positional or finite memory). From their results, one can derive the finite memory determinacy of membership games for alternating Streett tree automata, resulting in a proof that is different from the standard proofs that construct finite memory strategies by analyzing the whole game graph. Intuitively, their construction starts from an arbitrary winning strategy, and turns it into a finite memory strategy by looking at each node at the possible plays that arrive at this node, and basing the strategy decision only on this information.

The construction presented in [39] yields a nondeterministic automaton with $2^{O(nd \log(n))}$ states, which is slightly better than the result from Theorem 3.10. The advantage of the construction that we presented here is that it is modular and that the complexity of certain steps is encapsulated in separate theorems on games and ω -automata. The presentation is based on the transparent complementation construction for nondeterministic tree automata that was introduced by Gurevich and Harrington in [22], and is very nicely presented in [56].

We have presented the simulation construction for parity tree automata. The same techniques also work for Rabin conditions with the same complexities (where the number d of priorities is replaced by the number of Rabin pairs): Rabin conditions also admit positional strategies for Eva, and the size of the deterministic automaton on infinite words that is constructed in the proof of Lemma 3.9 is of the same order. However, instead of determinising a nondeterministic Büchi word automaton, one needs to determinise a nondeterministic Streett word automaton, using a construction from [49].

When starting from an alternating Streett tree automaton, then the situation changes, because winning strategies for Eva in Streett games require memory, in general. If the number of pairs in the Streett condition is d , then one has to consider strategies using memory of size up to $d!$. The ω -automaton constructed in the proof of Lemma 3.9 is therefore doubly exponential in d (but only singly exponential in n , as before).

An extension of this simulation result to two-way alternating parity tree automata that can additionally move up and down in the tree, is presented in [57].

3.4. Unambiguous and guidable automata. The simulation result from Theorem 3.10 shows that nondeterministic and alternating automata over infinite trees have the same expressive power. Concerning deterministic automata, there are two possible notions of a deterministic transition relation. For top-down determinism, it is required that for each state q and each letter a , there is at most one transition of the form (q, a, q_1, q_2) . Bottom-up determinism requires that for all states q_1, q_2 and all letters a , there is at most one transition of the form (q, a, q_1, q_2) . In [33] it is shown that each regular tree language can be accepted by an automaton with a bottom-up deterministic transition relation. The construction is based on the fact that the run on the nodes up to depth ℓ can be encoded on the nodes at depth $\ell + 1$. However, besides a deterministic transition relation, a deterministic automaton also needs a unique starting point in order to produce a unique run on an input. On infinite trees, there is no unique starting point in the bottom-up view, and therefore the result from [33] does not provide an automaton

with a unique run on each input (in fact, the automaton constructed in [33] has at least as many runs as the given one).

On the other hand side, top-down deterministic automata are easily seen to be weaker than nondeterministic ones (they accept precisely the path-closed languages as shown in [59] for finite trees; the result can easily be adapted to infinite trees). This raises the question, whether there are other subclasses of nondeterministic automata with good properties and full expressive power. A natural candidate for such a class are unambiguous automata, which have at most one accepting run for each input. On finite words, there are more efficient algorithms for equivalence and inclusion testing [52] of unambiguous automata than for the class of all nondeterministic automata. This has been generalised to automata on finite trees in [51]. It is also known that on infinite words, each nondeterministic Büchi automaton has an equivalent unambiguous one [2], while deterministic Büchi automata are strictly weaker (see also Chapter 6 of this work).

For automata on infinite trees, we obtain a negative result: there are regular languages that cannot be accepted by an unambiguous automaton. This fact is already witnessed by the very simple language of all trees containing at least one b :

$$T_{\exists b} = \{t \in \mathcal{T}_{\{a,b\}}^\omega \mid \text{there exists } u \in \{0, 1\}^*: t(u) = b\}.$$

The proof that this language cannot be accepted by any unambiguous tree automaton, is based on the following idea: a tree t from $T_{\exists b}$ describes a non-empty set of tree nodes, namely those labelled b . An accepting run has to find such a position, witnessing the membership of t in $T_{\exists b}$. For each run, one can describe in MSO a unique such position that is chosen by the run. If there is only a single accepting run, then one can define an MSO formula that chooses for each non-empty set of nodes exactly one position in this set. This contradicts a result on the undefinability of choice functions by MSO formulas over the infinite binary tree [23], [8], and [9]. We obtain the following theorem:

Theorem 3.12 ([9]). *There is no unambiguous parity tree automaton accepting the language $T_{\exists b}$.*

The concept of guidable automata was introduced in [14]. Intuitively, a guidable automaton \mathcal{A} can simulate each automaton \mathcal{B} accepting the same language: the choices of transitions for building an accepting run of \mathcal{B} can be used to deterministically build an accepting run of \mathcal{A} . In this sense, \mathcal{B} acts as a guide for \mathcal{A} .

Formally, $\mathcal{A} = (Q_{\mathcal{A}}, A, \iota_{\mathcal{A}}, \Delta_{\mathcal{A}}, c_{\mathcal{A}})$ can be guided by $\mathcal{B} = (Q_{\mathcal{B}}, A, \iota_{\mathcal{B}}, \Delta_{\mathcal{B}}, c_{\mathcal{B}})$ if there is a mapping $g: Q_{\mathcal{A}} \times \Delta_{\mathcal{B}} \rightarrow \Delta_{\mathcal{A}}$ with the following properties:

- $g(p, (q, a, q', q'')) = (p, a, p', p'')$ for some $p', p'' \in Q_{\mathcal{A}}$;
- for every accepting run ρ of \mathcal{B} over a tree t , $g(\rho)$ is an accepting run of \mathcal{A} over t , where $g(\rho) = \rho'$ is the unique run such that $\rho'(\varepsilon) = \iota_{\mathcal{A}}$, and for all $u \in \{0, 1\}^*$:

$$(\rho'(u), t(u), \rho'(u0), \rho'(u1)) = g(\rho'(u), (\rho(u), t(u), \rho(u0), \rho(u1))).$$

The first condition ensures that the transition chosen by g is compatible with the given state of \mathcal{A} and the input letter of the given transition of \mathcal{B} . The second condition states that accepting runs of \mathcal{B} are translated into accepting runs of \mathcal{A} when applying g . If such a mapping g exists, then we say that (\mathcal{B}, g) guides \mathcal{A} .

Obviously, these conditions can only be satisfied if $T(\mathcal{B}) \subseteq T(\mathcal{A})$. We say that \mathcal{A} is *guidable* if it can be guided by every \mathcal{B} such that $T(\mathcal{B}) = T(\mathcal{A})$. In fact, we could also require that \mathcal{A} can be guided by every \mathcal{B} such that $T(\mathcal{B}) \subseteq T(\mathcal{A})$ (see [31] for a proof of the equivalence of the two definitions).

If (\mathcal{B}, g) guides \mathcal{A} , then the mapping g can also be seen as defining a simulation relation between tree automata (see [1] for simulation relations on automata for finite trees) with an additional fairness constraint for the acceptance condition (see [24]). In that sense, a guidable automaton can simulate every equivalent automaton. It is not difficult to see that the concept of guidable automata is orthogonal to the concept of unambiguity (see [31]). And as opposed to unambiguous tree automata, we obtain a positive result for guidable tree automata. It can be shown that the complementation procedure for nondeterministic tree automata (obtained by applying the constructions from Theorem 3.7 and Theorem 3.10 starting from a nondeterministic automaton), yields a guidable automaton.

Theorem 3.13 ([14]). *For each regular tree language T , there exists a guidable parity tree automaton that can be constructed from a given parity tree automaton for T .*

Concerning the complexity, we obtain a doubly exponential construction when starting from an automaton for T because we have to apply the complementation construction twice. One can show that this blow-up cannot be avoided, in general. We refer the reader to [31] for more details on guidable automata.

4. Decision problems

In this section we consider decision problems for automata on infinite trees. The most fundamental one is the emptiness problem, that is, to decide for a given tree automaton whether its language is empty. This problem plays a central role when using tree automata as a tool for decision procedures in logic (see § 5). Furthermore, other problems such as inclusion or equality of regular tree languages can be reduced to the emptiness problem by using constructions for the Boolean operations.

A transparent solution for the emptiness problem that we are going to explain in the following, is using games similar to the membership games presented in § 3.1. Recall that in the membership game the aim of Eva is to prove that there exists an accepting run of the automaton on the given tree, while Adam tries to identify a path witnessing that the run is not accepting. The resulting game graph is infinite because it depends on the given tree t .

The emptiness game is similar to the membership game, but now Eva's aim is to prove that there exists a tree and an accepting run on this tree, while Adam, as before, tries to identify a non-accepting path. Hence, we basically obtain the emptiness game by removing the reference to the given tree t from the membership game, and leave the choices of the labels to Eva. However, when constructing a tree from a strategy of Eva, the choices of the labels have to be consistent, meaning that if two plays according to the strategy arrive at the same node of the tree, then Eva has to make the same choices

of the label. For this reason, the emptiness game (in its simple form) only works for nondeterministic automata. Here it is guaranteed that for a fixed strategy of Eva, only one play arrives at each node of the tree.

We now give the formal definition of the *emptiness game* $\mathcal{G}_{\mathcal{A}}$ for a nondeterministic parity tree automaton $\mathcal{A} = (Q, A, \iota, \Delta, c)$. We note that the same principle works for all other types of acceptance conditions, by simply adapting the winning condition of the resulting game accordingly.

The set of vertices of $\mathcal{G}_{\mathcal{A}}$ is $Q \cup \Delta$, that is, it consists of the states and the transitions of \mathcal{A} . The states are the positions of Eva, and the transitions are the positions of Adam. The initial position of the game is the initial state ι . The moves of the game are described below:

- from a state q , Eva can move to all transitions of the form $(q, a, q_0, \dots, q_{k-1})$ with q as source state;
- from a transition $(q, a, q_0, \dots, q_{k-1})$, Adam can move to one of the states q_0, \dots, q_{k-1} . This corresponds to the choice of a direction $i \in [k]$.

The priority function of $\mathcal{G}_{\mathcal{A}}$ corresponds to the priority function of \mathcal{A} extended to transitions by setting $c(q, a, q_0, \dots, q_{k-1}) = c(q)$. We can now relate the existence of winning strategies in $\mathcal{G}_{\mathcal{A}}$ to the emptiness of $T(\mathcal{A})$ (see [62] and [56]):

Lemma 4.1. *Eva has a winning strategy in the emptiness game $\mathcal{G}_{\mathcal{A}}$ if and only if $T(\mathcal{A}) \neq \emptyset$.*

The size of the parity game $\mathcal{G}_{\mathcal{A}}$ (number of vertices plus number of edges) is linear in the size of \mathcal{A} , where the size $|\mathcal{A}|$ of \mathcal{A} is defined as $|Q| + |\Delta|(k + 2)$ because each transition contains $k + 2$ entries. In combination with Theorem 3.2, we obtain the decidability and a complexity result for the emptiness problem of parity tree automata. The decidability of the emptiness problem for Rabin tree automata was already shown in [47].

Theorem 4.2. *The emptiness problem for parity tree automata is in $NP \cap co-NP$. If the automaton has d priorities, then the emptiness problem can be solved in time $\mathcal{O}(|\mathcal{A}|^d)$ or in time $|\mathcal{A}|^{\mathcal{O}(\log(d))}$.*

As a consequence, we obtain that decision problems that can be reduced to the emptiness problem of nondeterministic parity tree automata are also decidable.

Corollary 4.3. *The inclusion problem for regular tree languages given by alternating parity tree automata can be solved in exponential time.*

Proof. Given two alternating parity tree automata \mathcal{A} and \mathcal{B} , it is possible to test the condition $T(\mathcal{A}) \subseteq T(\mathcal{B})$ by combining \mathcal{A} and $\tilde{\mathcal{B}}$ into a parity tree automaton accepting $T(\mathcal{A}) \cap (\mathcal{J}_{\mathcal{A},k}^\omega \setminus T(\mathcal{B}))$, whose size is the sum of the sizes of \mathcal{A} and \mathcal{B} . Then we apply Theorem 3.10, and obtain a nondeterministic parity tree automaton with exponentially many states and polynomially many priorities. The language of this automaton is empty if and only if $T(\mathcal{A}) \subseteq T(\mathcal{B})$. Since the algorithm for solving the emptiness problem is

only exponential in the number of priorities, the combined procedure runs in singly exponential time. \square

Since already the universality problem, that is, the question whether a given tree language contains all trees, is complete for exponential time for nondeterministic automata on finite trees [51], there is no hope to improve Corollary 4.3 for simpler acceptance conditions or nondeterministic automata.

The emptiness game and the result on the positional determinacy of parity games, can be used to show the existence of a certain kind of simple trees in each non-empty regular tree language. These trees are called regular because they can be generated by a deterministic finite automaton in the following sense: a tree $t \in \mathcal{T}_{A,k}^\omega$ is called *regular* if there is a deterministic finite automaton $(S, [k], s_0, \eta, \lambda)$ with state set S , input alphabet $[k]$, initial state s_0 , transition function η , and output function $\lambda: S \rightarrow A$, such that for each $u \in [k]^*$, the label $t(u)$ of u in t is the same as the output $\lambda(\eta(u))$ computed by the automaton. Here, $\eta(u)$ denotes the state reached by the automaton after reading u .

We can also view the deterministic automaton from the above definition as a finite graph of out-degree k , whose vertices are labelled by symbols from A (via λ). The tree it generates is simply the unfolding of this graph starting from the initial state s_0 .

It is not difficult to see that regular trees are exactly those that have only finitely many non-isomorphic subtrees: a tree with this property can be generated by a finite automaton whose states correspond to the different subtrees it contains. Vice versa, a tree that is generated by a finite automaton can only contain as many non-isomorphic subtrees as the number of states of the automaton.

From a positional strategy for Eva in the emptiness game \mathcal{G}_A , we can construct a regular tree in the language of A . The existence of a regular tree inside non-empty regular tree languages was already shown in [47]. The statement on the “size” of such a tree in the following theorem is an immediate consequence of positional determinacy of parity games.

Theorem 4.4 ([47]). *Every non-empty regular tree language T contains a regular tree. If T is accepted by a nondeterministic parity tree automaton \mathcal{A} , then a regular tree in T can be generated by an automaton with as most as many states as \mathcal{A} .*

Proof. A positional strategy σ_E for Eva chooses a transition $(q, a, q_0, \dots, q_{k-1})$ for each state q . We specify the output function of the automaton generating the tree as $\lambda(q) = a$, and the transition function by $\eta(q, i) = q_i$ for each $i \in [k]$. \square

The following theorem is an easy consequence of Theorem 4.4 and the closure of regular tree languages under difference.

Corollary 4.5. *Two regular tree languages are equal if and only if they contain the same regular trees.*

Regular trees can be viewed as a simple class of infinite trees that have a finite representation. Given an alternating tree automaton \mathcal{A} , and such a regular tree t , we might want to know whether the tree is accepted by \mathcal{A} . This is called the membership problem for regular trees. In §3.1, we have introduced the membership game $\mathcal{G}_{\mathcal{A},t}$,

characterising the membership of t in the language $T(\mathcal{A})$. For general trees t , the arena of $\mathcal{G}_{\mathcal{A},t}$ is infinite. For a regular tree t , the arena can be made finite, by identifying nodes of the tree that correspond to the same state of the finite automaton generating t . The size of the resulting game is a product of $|\mathcal{A}|$ and $|t|$, where the size $|\mathcal{A}|$ of \mathcal{A} is the sum of the number of states and the size of the formulas occurring in the transition function of \mathcal{A} , and the size $|t|$ of t is the size of the automaton representing t . We obtain the following theorem as a consequence of the above explanations and Theorem 3.2.

Theorem 4.6. *The membership problem for an alternating parity tree automaton \mathcal{A} with d priorities, and a regular tree t can be solved in time $\mathcal{O}((|\mathcal{A}| \cdot |t|)^d)$ or in time $(|\mathcal{A}| \cdot |t|)^{\mathcal{O}(\log(d))}$.*

We finish this section by briefly discussing the *parity index problem* for regular tree languages: given a regular tree language T , and a range i, \dots, j of priorities, decide if T can be accepted by a parity automaton using the interval i, \dots, j of priorities. There are two immediate variations of this problem, depending on whether we want the resulting parity automaton to be nondeterministic or alternating. For both types of automata, the hierarchy induced by growing ranges of priorities is known to be strict (see [40], [4] and [30]). However, deciding the levels of the respective hierarchies is still an open problem. In the following, we briefly explain some results on special cases for the nondeterministic parity index problem: given a regular tree language, and a priority range i, \dots, j , can T be accepted by a nondeterministic parity tree automaton with priorities i, \dots, j ?

If the given language is of the form $\text{Path}(L)$ for an ω -language L (see § 2.1), then we can use Theorem 2.3, and the fact that for deterministic parity word automata the index problem is decidable [41] and [11]. This decidability result can be generalised to the case where the given tree language can be accepted by a deterministic tree automaton [42]. The lowest level of the hierarchy, the case where $i = j = 0$, is known to be decidable for arbitrary regular tree languages as input [61]. If T is given by an alternating co-Büchi tree automaton, it is decidable whether it can be accepted by a nondeterministic Büchi automaton [13] (this result is rephrased in terms of logic in Theorem 5.4 in § 5).

Finally, an approach to tackle the general problem has been presented in [14], where the nondeterministic parity index problem is reduced to the universality problem for so called distance parity tree automata, but the decidability of the latter problem remains open.

5. Applications in logic

In this section, we apply the results on automata on infinite trees to solve logical decision problems. The original motivation for introducing this kind of automata was the decision problem for monadic second-order logic (MSO) of two successor functions, that is, of the infinite binary tree [45]. In § 5.1 we present Rabin's result on the equivalence

of MSO and tree automata, and also give a characterisation of weak monadic second-order logic (WMSO) in terms of alternating automata. The equivalence of MSO and automata can be used to decide satisfiability of MSO over the infinite binary tree. We show how to use this result to solve the synthesis problem for logics describing properties of infinite sequences. Intuitively, this problem is about automatically synthesising a program all of whose computations satisfy a given specification. This problem can be translated to a problem over infinite trees, where the paths of the tree correspond to the possible computations.

In § 5.2 we briefly explain the relation between the modal μ -calculus, an extension of classical modal logic by fixed point operators, and tree automata.

5.1. Monadic second-order logic. Before we come to the relation between automata and logic, we first fix some basic terminology. We use *monadic second-order logic* (MSO) over relational structures with the standard syntax and semantics (see e.g., [16] for a detailed presentation). MSO formulas use first-order variables, which are interpreted by elements of the structure, and monadic second-order variables, which are interpreted as sets of elements. Small letters (e.g., x, y) denote first-order variables, and capital letters (e.g., X, Y) denote monadic second-order variables. First-order logic (FOL) is the fragment of MSO that does not use set quantifications.

We view the unlabelled k -ary tree as the relational structure $t_k = ([k]^*, E_0, \dots, E_{k-1})$, where the E_i are binary symbols interpreted as $\{(w, w.i) \mid w \in [k]^*\}$. MSO logic over t_2 is also referred to as S2S (second-order theory of two successors).

An MSO formula $\varphi(X_1, \dots, X_n)$ with n free set variables, defines a language of trees over the alphabet $[2]^n$. The labelling is used to code the interpretations of the free set variables. For this purpose, we define for given sets $U_1, \dots, U_n \subseteq [k]^*$ the characteristic tree $t[U_1, \dots, U_n]$, in which each node u is labelled by the tuple $(b_1, \dots, b_n) \in [2]^n$, with $b_i = 1$ if $u \in U_i$ and $b_i = 0$ otherwise.

The tree language $T(\varphi) \subseteq \mathcal{T}_{[2]^n, k}^\omega$ of φ is

$$T(\varphi) = \{t[U_1, \dots, U_n] \mid t_k \models \varphi[U_1, \dots, U_n]\},$$

where $\varphi[U_1, \dots, U_n]$ means that each variable X_i is interpreted by the set U_i . We also write $t[U_1, \dots, U_n] \models \varphi$ if $t[U_1, \dots, U_n] \in T(\varphi)$.

To simplify notation, we state the results below for binary trees, but everything remains valid for the more general case of k -ary trees.

The key result from [45] for showing the decidability of MSO over t_2 , is the correspondence between tree languages definable by MSO and by automata. Based on this, the decidability problem of the logic can be reduced to the emptiness problem of automata.

Theorem 5.1 ([45]). *A tree language $T \subseteq \mathcal{T}_{[2]^n}^\omega$ is definable in MSO if and only if it is regular. The translation between formulas and automata is effective in both directions.*

The translation from formulas to automata easily follows from the closure properties of automata. The translation in the other direction uses the standard technique of

describing an accepting run of the automaton using sets to encode the states in the run (see e.g., [56]).

There is a variant of MSO that only differs in the way set quantifications are interpreted: formulas of *weak monadic second-order logic* (WMSO) have the same syntax as MSO formulas, but the set quantifiers only range over finite sets. Free variables of a WMSO formula are still interpreted by arbitrary (finite and infinite) sets.

Note that at first glance, these two logics are incomparable. For example, over t_2 the sentence $\forall X \exists x: x \notin X$ is false as an MSO sentence, but it is true as a WMSO sentence. However, we can translate WMSO formulas into MSO formulas because finiteness of sets can be expressed in MSO over t_2 : a set of nodes of t_2 is finite if and only if there exists a cut through the tree (a set of nodes pairwise incomparable with respect to the prefix relation that intersects every path) such that all elements of the set are above the cut.

A translation of MSO formulas into equivalent WMSO formulas is not possible, in general. In [46] Rabin has shown that a language can be defined in WMSO if and only if the language and its complement can be accepted by a Büchi tree automaton.³ Later, building on the result of Rabin, a characterisation of WMSO definable languages in terms of weak alternating tree automata has been given [37] (see § 2 for the definition of weak alternating automata).

Theorem 5.2 ([46] and [37]). *For a language $T \subseteq \mathcal{T}_{[2]^n}^\omega$ the following are equivalent:*

1. T is definable in WMSO;
2. T and $\mathcal{T}_{[2]^n}^\omega \setminus T$ can be accepted by a nondeterministic Büchi tree automaton;
3. T can be accepted by a weak alternating tree automaton.

In the following, we discuss the main ingredients for the proof of Theorem 5.2. The implication from (3) to (2) easily follows from the results presented in § 3: weak alternating tree automata are special cases of alternating Büchi automata, and therefore can be turned into nondeterministic Büchi automata according to Theorem 3.8. Furthermore, weak alternating tree automata are closed by dualisation and thus we can also construct a nondeterministic Büchi automaton for the complement language using Theorem 3.7.

For the implication from (1) to (3), one can give an inductive translation of WMSO formulas into weak alternating tree automata [37]. It is easy to see that atomic formulas are equivalent to weak alternating tree automata. Furthermore, weak alternating tree automata are closed under union, intersection, and complement (see § 3). The only remaining operation that we need to consider, is the one of existential quantification over finite sets, which can be done by the operation of finite projection: let A and B be two alphabets with $B \subseteq A$, and let $h: A \rightarrow B$ be a mapping that preserves the labels of B , i.e., with $h(a) = a$ for all $a \in B$. As usual, we apply h to a tree $t \in \mathcal{T}_{A,k}^\omega$ by applying it to the label of each node. The finite projection of a language $T \subseteq \mathcal{T}_{A,k}^\omega$ consists of the set of all trees that are obtained by applying h to trees t in T that only

³ Rabin used the notion “special automaton.”

have finitely many labels in $A \setminus B$:

$$h_{\text{fin}}(T) = \{h(t) \mid t \in T \text{ and } t(x) \in A \setminus B \text{ for only finitely many } x\}.$$

The operation of finite projection can be used to capture the operation of weak existential quantification. Therefore, the following proposition is sufficient to complete the transformation of WMSO formulas into weak alternating tree automata.

Proposition 5.3 ([37]). *The class of languages definable by weak alternating tree automata is closed under the operation of finite projection.*

Proof. The idea for the construction is as follows. If there are only finitely many nodes with a label from $A \setminus B$, then there is a finite initial part of the tree containing all these nodes. We use a simple subset construction that collects different possible computations along the same branch in a run, and at some point guesses that below the current node no labels from $A \setminus B$ occur any longer. From these nodes, we run the original automaton \mathcal{A} starting from all states collected in the subset automaton. \square

To finish the proof of Theorem 5.2, it remains to show the implication from (2) to (1). Assume that a given language $T \subseteq \mathcal{T}_{A,k}^\omega$ is such that T and its complement can be accepted by Büchi tree automata \mathcal{A} and \mathcal{B} , respectively. An accepting run of a Büchi automaton has to visit infinitely many accepting states on each path. This means that we can find infinitely many pairwise disjoint cuts in the run (a cut is an inclusion minimal set of nodes that intersects each path of the tree), such that for each cut, all the elements are labelled by accepting states. We call these accepting cuts. Equivalently, we can find accepting cuts at arbitrary depth in the run. The key idea is to show that there is a bound m , such that a tree t is accepted by \mathcal{A} if and only if there are initial parts of runs that have m accepting cuts at arbitrary depth (we omit the proof, which can be found in [46]).

We now explain the WMSO formula that formally defines this property. For simplicity, we use some auxiliary formulas that can easily be defined in WMSO. We write $Fr(X)$ for the frontier of X , i.e., the set of maximal elements with respect to the prefix ordering of X . Furthermore, we allow quantification over partial runs ρ : we write $\rho \in Run_{\mathcal{A}}(X)$ to indicate that X is a prefix closed set, and ρ is a partial run of \mathcal{A} on X , that is, a labelling $\rho: X \rightarrow Q_{\mathcal{A}}$ that starts in an initial state, and is consistent with the transitions. This can be written in standard WMSO by using several sets for coding ρ .

The formula φ_m is then defined as follows, where $X_1, Y_1, \dots, X_m, Y_m$ should be thought of as an increasing sequence of prefix closed sets:

$$\forall X_1 \exists Y_1 \dots \forall X_m \exists Y_m \\ \exists \rho \in Run_{\mathcal{A}}(Y_m) \left(\bigwedge_{i=1}^m [X_i \subseteq Y_i \wedge \forall y \in Fr(Y_i) (\rho(y) \in F_{\mathcal{A}})] \right).$$

Note that the formula φ_m has n free variables coding the labelling of the input tree. These free variables are implicitly used in the statement $\rho \in Run_{\mathcal{A}}(Y_m)$. The formal proof that this defines the language $T(\mathcal{A})$ can be found in [46]. This finishes the discussion of the proof of Theorem 5.2.

It is an open problem whether it is decidable for a given MSO formula or parity tree automaton, whether there exists an equivalent WMSO formula. In view of Theorem 5.2(2), this question is related to the parity index problem that is discussed at the end of § 3. A partial result has been obtained for languages given by Büchi tree automata:

Theorem 5.4 ([13]). *Given a Büchi tree automaton A , it is decidable whether $T(A)$ is definable in WMSO.*

The connection between automata and logic, as presented in Theorems 5.1 and 5.2, allows us to develop decision procedures for the logic, based on algorithms for automata. The satisfiability problem for MSO over t_2 , i.e., the question whether for a given MSO formula there is an interpretation of the free variables, such that the formula is satisfied in t_2 , is reduced to the emptiness problem for tree automata. The latter is decidable, as we have seen in § 4 (concerning the complexity, already over finite words the translation of MSO into automata involves a non-elementary blow-up [53]).

Theorem 5.5 ([45]). *The satisfiability problem for MSO over t_2 is decidable.*

If a formula $\varphi(X_1, \dots, X_n)$ is satisfiable, then the equivalent automaton that exists according to Theorem 5.1 accepts a non-empty language and thus a regular tree by Theorem 4.4. As a consequence, we obtain the following result.

Corollary 5.6 ([47]). *Let $\varphi(X_1, \dots, X_n)$ be a satisfiable MSO formula over t_2 . There are regular sets $U_1, \dots, U_n \subseteq [2]^*$ such that $t[U_1, \dots, U_n] \models \varphi$.*

As an application of Theorem 5.5, we now discuss the problem of synthesising a so-called “reactive module,” which reads a stream $a_1 a_2 \dots$ of input symbols, and synchronously produces a stream $b_1 b_2 \dots$ of output symbols, such that the combined sequence $(a_1, b_1)(a_2, b_2) \dots$ satisfies a given specification. The task is to automatically synthesise such a reactive module from the specification that is given in some formal system. More precisely, if A and B are the sets of input and output symbols, respectively, then a specification is a set $L \subseteq (A \times B)^\omega$ of allowed computations. A realisation of such a specification L , is a function $f: A^+ \rightarrow B$ such that each sequence of the form $(a_1, f(a_1))(a_2, f(a_1 a_2))(a_3, f(a_1 a_2 a_3)) \dots$ is in L . The synthesis problem is to decide for a given specification whether it is realisable, and to construct a realising function if possible.

It is clear that the difficulty of this problem depends on how complex the allowed specifications are. We focus here on MSO as specification formalism. In this setting, the sets of input and output symbols are of the form $[2]^m$ and $[2]^n$, respectively. The specification is given as an MSO formula over infinite words, that is, over the structure of the natural numbers with successor function. The formula is of the form $\varphi(X_1, \dots, X_m, Y_1, \dots, Y_n)$, where the variables X_1, \dots, X_m code the input sequence, and Y_1, \dots, Y_n code the output sequence.

We are interested in functions of the form $f: ([2]^m)^+ \rightarrow [2]^n$. Such a function can easily be represented by a k -ary $[2]^n$ -labelled tree t_f with $k = 2^m$. Each node

x of t_f naturally corresponds to an input sequence, since elements of $[k]^*$ correspond to sequences over $[2]^m$. The value $t_f(x)$ is then the value of f assigned to the input sequence corresponding to x . For the node ε , we pick some arbitrary label. Based on this idea, one can solve the synthesis problem by translating φ into an MSO formula over t_2 that defines precisely the trees of the form t_f for some f realising φ [47]. A related approach using games has been used by Büchi and Landweber in [6].

Theorem 5.7 ([6] and [47]). *The synthesis problem for MSO specifications is decidable, and can be reduced to the satisfiability problem for MSO over the infinite binary tree.*

The solution of the synthesis problem that we have sketched above, requires a translation of MSO formulas into tree automata. This translation is of non-elementary complexity, in general. It is also clear that the satisfiability problem for MSO can easily be encoded in the realisability problem, hence this non-elementary complexity cannot be avoided. Therefore, it is worth to consider different specification formalisms that admit solutions of the above problems with better complexity.

Pnueli and Rosner [44] have shown that synthesis for formulas in the linear temporal logic LTL can be done in doubly exponential time. The basic idea is to construct a deterministic ω -automaton that accepts the admissible behaviours according to the specification. This deterministic automaton is used as tree automaton by running it over all the branches of a tree of suitable branching degree: the direction in the tree codes the inputs, and the labels of the nodes the outputs (as in the description before Theorem 5.7). A regular tree (see § 4) in the language of the resulting automaton provides a finite state program realising the specification. Since determinisation of ω -automata involves complex constructions, some alternative approaches have been developed that avoid explicit determinisation constructions [29] and [20].

We finish this section on MSO, by mentioning some further results that are related to, or extend Rabin's decidability result for the binary tree.

A result from [15] shows that the unfolding operation for graphs preserves decidability of the MSO theory. The unfolding of a graph G from a vertex v is a graph whose vertices are paths in G starting from v . The edges correspond to path extensions by one edge in G , and the node labels are inherited from the last element of the path. Note that the infinite binary tree can easily be obtained by unfolding a graph consisting of a single node and two self-loops, and therefore Theorem 5.5 can easily be obtained from the result in [15].

A more general operation that can be applied to all structures and not only graphs, is the iteration of a structure that arranges copies of the structure in a tree-like fashion. Details on the iteration operation, its properties, and further references can be found in [3]. The unfolding of a graph can be reconstructed from its iteration by an MSO interpretation. An interpretation defines a new structure inside a given one by means of logical formulas, MSO formulas in our case. The decidability of the MSO theory is preserved by this operation because we can translate each MSO formula over the new structure into an MSO formula over the old structure, by replacing each atomic formula

by its defining formula from the interpretation. Starting from the class of finite graphs, alternating application of the operations of unfolding and interpretation leads to the Caucal hierarchy [12].⁴ Since both operations, unfolding and interpretation preserve MSO decidability, all graphs in the Caucal hierarchy have a decidable MSO theory.

The method of interpretation mentioned above can also be used to obtain undecidability results. In [8] it is shown that in every extension of t_2 by a well-ordering (a total ordering without infinite decreasing chains) of its domain one can interpret t_2 extended with the length-lexicographic ordering of the nodes (nodes are ordered according to their length and nodes of equal length according to the lexicographic ordering). This latter extension of t_2 is known to have an undecidable MSO theory (see [55]), and therefore the MSO theory of every extension of t_2 by a well-ordering is undecidable.

5.2. Modal μ -calculus. The modal μ -calculus (μ -calculus for short) is an extension of propositional modal logic by fixed point operators. We describe in this section, how formulas of the μ -calculus can be translated into alternating automata, thus obtaining algorithms for solving the satisfiability, and the model checking problem for this logic. For a simple presentation within our framework of ordered trees, we only consider binary trees as models in this section. More general frameworks with arbitrary Kripke structures as models, are based on automata for unordered trees, and can be found, for example, in [21] and [58]. The underlying techniques, however, are the same in both settings.

Formulas of the μ -calculus are built from a set $P = \{p_1, \dots, p_n\}$ of atomic propositions, variables X, Y, \dots , Boolean combinations of formulas, modal formulas $\Diamond\varphi$ and $\Box\varphi$ for a formula φ , and fixed point formulas $\mu X.\varphi(X)$ and $\nu X.\varphi(X)$, where φ is a formula in which X only occurs positively (under an even number of negations).

As models, we consider binary trees over the alphabet $A = 2^P$. A formula defines a set of nodes in a tree $t \in \mathcal{T}_A^\omega$: the set of nodes at which the formula holds. Intuitively, a formula $\Diamond\varphi$ holds at some node if φ holds at least at one of the successors, and $\Box\varphi$ holds at some node if φ holds at all successors. To define the semantics of fixed point formulas, a formula $\varphi(X)$ with a free variable X can be seen as an operator that maps sets U of nodes (interpretations of X) to sets U' of nodes (the set of nodes at which $\varphi(X)$ holds with U as interpretation of X). If the variable X occurs only positively in $\varphi(X)$, then this operator is monotone in the sense that interpreting X by larger sets leads to larger sets of nodes at which $\varphi(X)$ is true. The Knaster–Tarski fixed point theorem yields that the operator defined by $\varphi(X)$ has a unique least and a unique greatest fixed point. These fixed points are defined by the formulas $\mu X.\varphi(X)$ (least fixed point) and $\nu X.\varphi(X)$ (greatest fixed point).

To formally define the set of nodes at which a formula holds, we assume that an interpretation f of the variables is given, that is, f maps each variable of φ to a set of nodes of t . Then the semantics of the formulas is defined inductively as follows:

⁴ In the original formulation, inverse rational substitutions are used instead of interpretations, but the resulting classes are the same [10].

- $\llbracket p_i \rrbracket_t^f = \{u \in [2]^* \mid p_i \in t(u)\};$
- $\llbracket X \rrbracket_t^f = f(X);$
- $\llbracket \neg\varphi \rrbracket_t^f = [2]^* \setminus \llbracket \varphi \rrbracket_t^f$ (and similarly for the other Boolean operations);
- $\llbracket \diamond\varphi \rrbracket_t^f = \{u \in [2]^* \mid u.i \in \llbracket \varphi \rrbracket_t^f \text{ for some } i \in [2]\};$
- $\llbracket \square\varphi \rrbracket_t^f = \{u \in [2]^* \mid u.i \in \llbracket \varphi \rrbracket_t^f \text{ for all } i \in [2]\};$
- $\llbracket \mu X.\varphi(X) \rrbracket_t^f = \bigcap \{U \subseteq [2]^* \mid U = \llbracket \varphi \rrbracket_t^{f[X \mapsto U]}\},$ where $f[X \mapsto U]$ is the interpretation of variables that interprets X as U , and all other variables Y as $f(Y)$;
- $\llbracket \nu X.\varphi(X) \rrbracket_t^f = \bigcup \{U \subseteq [2]^* \mid U = \llbracket \varphi \rrbracket_t^{f[X \mapsto U]}\}.$

Our aim is to translate such formulas into alternating parity tree automata. We first observe that the negation can be pushed in front of atomic formulas by the ordinary De Morgan laws, and the following rules: $\neg \diamond \varphi \equiv \square \neg \varphi$ and $\neg \mu X.\varphi(X) \equiv \nu X.\neg\varphi[\neg X/X]$, where $\varphi[\neg X/X]$ denotes the formula φ in which $\neg X$ is substituted for each free occurrence of X . These equivalences directly follow from the definition of the semantics.

The structure of the μ -calculus formulas is already very close to the transitions of an alternating automaton: we can handle atomic formulas by checking the labels of the tree, the operators \square and \diamond by sending states to all successors, or nondeterministically to one successor, and disjunction and conjunctions by the Boolean combinations in the transition function. So the idea is to take the sub-formulas of a given formula as states of the alternating automaton. The transitions are descending into the formulas according to the above rules.

The fixed point formulas are treated as follows: assume that we arrive at a fixed point variable X inside a formula $\mu X.\varphi(X)$. Intuitively, this means that we want to prove the formula correct by claiming that the current node is in the set X . The set X is defined by the formula $\mu X.\varphi(X)$, that is, we jump back to the definition of the fixed point formula. However, if we do this infinitely often without leaving the formula $\mu X.\varphi(X)$ (through fixed point variables that are defined outside the formula $\mu X.\varphi(X)$), then we are not able to prove that the node indeed has to be in the least fixed point defined by $\varphi(X)$. Thus, the automaton should reject in this case. We achieve this by assigning odd priorities to states corresponding to least fixed point formulas.

We proceed similarly for greatest fixed point formulas, assigning even priorities to the states corresponding to such formulas.

For the formal definition, according to the above description, the automaton only moves in the tree for \diamond - and \square -formulas. For this reason, we allow extended transition functions of the form $\delta: Q \times A \rightarrow \mathcal{B}^+((Q \times [k]) \cup Q)$, that is, the automaton can also change the state without moving in the tree (ϵ -transitions). When reaching an atom q in such a transition formula in the membership game, then the game proceeds in state q while remaining in the same node of the tree. This modification does not change any of the results presented, nor does it require new techniques.

Assuming that all the fixed point variables are different, the alternating parity tree automaton \mathcal{A}_ψ for a μ -calculus sentence ψ (without free variables) has as state set the

set of all sub-formulas of ψ . We let $\langle \psi \rangle$ denote the state corresponding to a formula ψ . The transition function δ is defined as follows:

- $\delta(\langle \lambda X.\varphi \rangle, a) = \langle \varphi \rangle$ for $\lambda \in \{\mu, \nu\}$;
- $\delta(\langle \diamond \varphi \rangle, a) = (\langle \varphi \rangle, 0) \vee (\langle \varphi \rangle, 1)$;
- $\delta(\langle \square \varphi \rangle, a) = (\langle \varphi \rangle, 0) \wedge (\langle \varphi \rangle, 1)$;
- $\delta(\langle \psi_1 \wedge \psi_2 \rangle, a) = \langle \psi_1 \rangle \wedge \langle \psi_2 \rangle$;
- $\delta(\langle \psi_1 \vee \psi_2 \rangle, a) = \langle \psi_1 \rangle \vee \langle \psi_2 \rangle$;
- $\delta(\langle p_i \rangle, a) = \mathbf{tt}$ if $p_i \in a$ and $\delta(\langle p_i \rangle, a) = \mathbf{ff}$ if $p_i \notin a$;
- $\delta(\langle X \rangle) = \langle \lambda X.\varphi(X) \rangle$ for the unique fixed point formula binding X .

The priority function is defined as explained above: least fixed point formulas are assigned odd priorities, and greatest fixed point formulas are assigned even priorities, where the outermost formulas are of highest priority and the innermost formulas of lowest priority.

The idea is illustrated in Figure 4 for the formula $\nu Y.\mu X.\diamond((p \wedge Y) \vee X)$ over a single proposition p . The picture shows the parse tree of the formula together with the back edges from the fixed point variables to their definitions. All arrows can be interpreted as ϵ -transitions of the automaton, except the arrow leaving the \diamond -operator. The $0 \vee 1$ next to this edge indicates that the automaton can nondeterministically choose to move to the left or the right successor in the tree. The two numbers next to the fixed point operators are the priorities, an even priority for the greatest fixed point formula, and an odd one for the least fixed point formula. The priority of the outer formula is 2, and thus higher than the one of the inner formula.

An analysis of this transition diagram yields that a tree is accepted if it contains a path on which p is true infinitely often: using the nondeterministic choice at the \diamond -state, the automaton guesses a path. It can cycle finitely often through the variable X , but in order to accept, it has to leave this cycle eventually. This means, it has to move to the conjunctive formula. At the corresponding node, p has to be true because of the sub-formula p , and the whole process is restarted via the variable Y , which jumps back to the outermost formula.

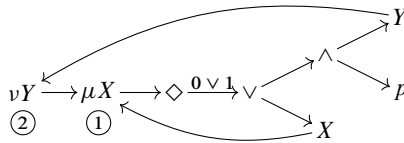


Figure 4. A graphical representation of the automaton for the formula $\nu Y.\mu X.\diamond((p \wedge Y) \vee X)$

We state the following theorem without proof.

Theorem 5.8 ([18]). *The alternating parity tree automaton \mathcal{A}_ψ accepts precisely those trees t such that ψ holds at the root of t .*

Using the results from § 4, we can now solve the satisfiability problem, and the model-checking problem for regular trees for the μ -calculus. The model-checking problem for regular trees is: given a μ -calculus formula φ , and a regular tree t , does φ hold at the root of t ?

The satisfiability problem reduces to the emptiness problem for alternating tree automata, which can be solved using Theorem 3.10 and Theorem 4.2.

Theorem 5.9 ([18]). *The satisfiability problem for the μ -calculus can be solved in exponential time.*

The model-checking problem for regular trees reduces to the membership problem for regular trees considered in Theorem 4.6. For the complexity, we have to estimate the number of priorities of the automaton \mathcal{A}_ψ . It turns out that we do not need new priorities for all fixed point formulas, but only for nested formulas of different type. The alternation depth of a formula is, roughly speaking, the number of nestings between least and greatest fixed point formulas.

Theorem 5.10 ([19]). *The model-checking problem on regular trees for the μ -calculus can be solved in time $\mathcal{O}((|\psi| \cdot |t|)^d)$ or in time $(|\psi| \cdot |t|)^{\mathcal{O}(\log(d))}$ where d is the alternation depth of the formula.*

References

- [1] P. A. Abdulla, A. Bouajjani, L. Holík, L. Kaati, and T. Vojnar, Computing simulations over tree automata. In *Tools and algorithms for the construction and analysis of systems* (C. R. Ramakrishnan and J. Rehof, eds.). Proceedings of the 14th International Conference (TACAS 2008) held as part of the Joint European Conferences on Theory and Practice of Software (ETAPS 2008) in Budapest, March 29–April 6, 2008. Lecture Notes in Computer Science, 4963. Springer, Berlin, 2008, 93–108. [MR 2503440](#) [Zbl 1134.68391](#) q.v. 285
- [2] A. Arnold, Rational ω -languages are non-ambiguous. *Theoret. Comput. Sci.* 26 (1983), no. 1–2, 221–223. [MR 0726919](#) [Zbl 0523.68069](#) q.v. 284
- [3] D. Berwanger and A. Blumensath, The monadic theory of tree-like structures. In *Automata, logics, and infinite games* (E. Grädel, W. Thomas, and T. Wilke, eds.). A guide to current research. Lecture Notes in Computer Science, 2500. Springer, Berlin, 2002, 285–301. [MR 2070747](#) [Zbl 1021.68051](#) q.v. 293
- [4] J. C. Bradfield, The modal mu-calculus alternation hierarchy is strict. *Theoret. Comput. Sci.* 195 (1998), no. 2, 133–153. [MR 1609327](#) [Zbl 0915.03017](#) q.v. 288
- [5] J. R. Büchi, On a decision method in restricted second order arithmetic. In *Logic, Methodology and Philosophy of Science* (E. Nagel, P. Suppes, and A. Tarski, eds.). Proceedings of the 1960 International Congress. Stanford University Press, Stanford, CA, 1962, 1–11. [MR 0183636](#) [Zbl 0147.25103](#) q.v. 267
- [6] J. R. Büchi and L. H. Landweber, Solving sequential conditions by finite-state strategies. *Trans. Amer. Math. Soc.* 138 (1969), 295–311. [MR 0280205](#) [Zbl 0182.02302](#) q.v. 293
- [7] C. S. Calude, S. Jain, B. Khoussainov, W. Li, and F. Stephan, Deciding parity games in quasipolynomial time. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing* (H. Hatami, P. McKenzie, and V. King, eds.). STOC '17. Held in

- Montreal, QC, June 19–23, 2017. Association for Computing Machinery, New York, 2017, 252–263. [MR 3678186](#) [Zbl 1369.68234](#) q.v. 275
- [8] A. Carayol and C. Löding, MSO on the infinite binary tree: choice and order. In *Computer Science Logic* (J. Duparc and T. A. Henzinger, eds.). Proceedings of the 21st International Workshop (CSL 2007), the 16th Annual Conference of the EACSL, held at the University of Lausanne, Lausanne, September 11–15, 2007. Lecture Notes in Computer Science, 4646. Springer, Berlin, 2007, 161–176. [MR 2540200](#) [Zbl 1179.03016](#) q.v. 284, 294
- [9] A. Carayol, C. Löding, D. Niwiński, and I. Walukiewicz, Choice functions and well-orderings over the infinite binary tree. *Cent. Eur. J. Math.* 8 (2010), no. 4, 662–682. [MR 2671218](#) [Zbl 1215.03054](#) q.v. 284
- [10] A. Carayol and S. Wöhrle, The Caucal hierarchy of infinite graphs in terms of logic and higher-order pushdown automata. In *FST TCS 2003: Foundations of software technology and theoretical computer science* (P. K. Pandya and J. Radhakrishnan, eds.). Proceedings of the 23rd Conference held in Mumbai, December 15–17, 2003. Lecture Notes in Computer Science, 2914. Springer, Berlin, 2003, 112–123. [MR 2093642](#) [Zbl 1205.03022](#) q.v. 294
- [11] O. Carton and R. Maceiras, Computing the Rabin index of a parity automaton. *Theor. Inform. Appl.* 33 (1999), no. 6, 495–505. [MR 1747513](#) [Zbl 0958.68089](#) q.v. 288
- [12] D. Caucal, On infinite terms having a decidable monadic theory. In *Mathematical foundations of computer science 2002* (K. Diks and W. Rytter, eds.). Papers from the 27th International Symposium (MFCS 2002) held in Warsaw, August 26–30, 2002. Lecture Notes in Computer Science, 2420. Springer, Berlin, 2002, 165–176. [MR 2064455](#) [Zbl 1014.68077](#) q.v. 294
- [13] T. Colcombet, D. Kuperberg, C. Löding, and M. Vanden Boom, Deciding the weak definability of Büchi definable tree languages. In *Computer science logic 2013* (S. Ronchi Della Rocca, ed.). LIPIcs. Leibniz International Proceedings in Informatics, 23. Schloss Dagstuhl. Leibniz-Zentrum für Informatik, Wadern, 2013, 215–230. [MR 3111742](#) [Zbl 1356.68129](#) q.v. 288, 292
- [14] T. Colcombet and C. Löding, The non-deterministic Mostowski hierarchy and distance-parity automata. In *Automata, languages and programming* (L. Aceto, I. Damgård, L. A. Goldberg, M. M. Halldórsson, A. Ingólfssdóttir, and I. Walukiewicz, eds.). Part II. Proceedings of the 35th International Colloquium (ICALP 2008) held in Reykjavik, July 7–11, 2008. Lecture Notes in Computer Science, 5126. Springer, Berlin, 2008, 398–409. [MR 2503604](#) [Zbl 1165.68038](#) q.v. 284, 285, 288
- [15] B. Courcelle and I. Walukiewicz, Monadic second-order logic, graph coverings and unfoldings of transition systems. *Ann. Pure Appl. Logic* 92 (1998), no. 1, 35–62. q.v. 293
- [16] H.-D. Ebbinghaus and J. Flum, *Finite model theory*. Second revised and enlarged edition. Springer Monographs in Mathematics. Springer, Berlin, 2006. [MR 2177676](#) [Zbl 1081.03026](#) q.v. 289
- [17] E. A. Emerson and C. S. Jutla, The complexity of tree automata and logics of programs (extended abstract). In *29th Annual Symposium on Foundations of Computer Science*. Held in White Plains, N.Y., October 24–26, 1988. IEEE Press, Los Alamitos, CA, 1988, 328–337. [IEEEExplore 21949](#) q.v. 275
- [18] E. A. Emerson and C. S. Jutla, Tree automata, mu-calculus and determinacy. In *Proceedings of the 32nd Annual Symposium of Foundations of Computer Science*. Held in San Juan, Puerto Rico, October 1–4, 1991. IEEE Press, Los Alamitos, CA, 368–377. [IEEEExplore 185392](#) q.v. 282, 296, 297

- [19] E. A. Emerson and C.-L. Lei, Efficient model checking in fragments of the propositional mu-calculus (extended abstract). In *Proceedings of the First Annual IEEE Symposium on Logic in Computer Science*. LICS 1986. Cambridge, MA, 1986. IEEE Press, Los Alamitos, CA, 1986, 267–278. q.v. [297](#)
- [20] E. Filiot, N. Jin, and J.-F. Raskin, An antichain algorithm for LTL realizability. In *Computer aided verification* (A. Bouajjani and O. Maler, eds.). Proceedings of the 21st international conference, CAV 2009, Grenoble, France, June 26–July 2, 2009. Lecture Notes in Computer Science, 5643. Springer, Berlin, 2009, 263–277. [Zbl 1242.68158](#) q.v. [293](#)
- [21] E. Grädel, W. Thomas, and T. Wilke (eds.), *Automata, logics, and infinite games*. A guide to current research. Lecture Notes in Computer Science, 2500. Springer, Berlin, 2002. [MR 2070731](#) [Zbl 1011.00037](#) q.v. [275](#), [276](#), [294](#)
- [22] Y. Gurevich and L. Harrington, Trees, automata, and games. In *Proceedings of the 14th annual ACM symposium on theory of computing*. (H. R. Lewis, B. B. Simons, W. A. Burkhard, and L. H. Landweber, eds.). Held in San Francisco, CA, May 5–7, 1982. Association for Computing Machinery, New York, 1982, 60–65. q.v. [276](#), [277](#), [280](#), [283](#)
- [23] Y. Gurevich and S. Shelah, Rabin’s uniformization problem. *J. Symbolic Logic* 48 (1983), no. 4, 1105–1119. [MR 0727798](#) [Zbl 0537.03007](#) q.v. [284](#)
- [24] T. A. Henzinger, O. Kupferman, and S. K. Rajamani, Fair simulation. *Inform. and Comput.* 173 (2002), no. 1, 64–81. [MR 1889125](#) [Zbl 1009.68071](#) q.v. [285](#)
- [25] M. Jurdzinski and R. Lazić, Succinct progress measures for solving parity games. *2017 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. Held in Reykjavík, Iceland, June 20–23, 2017. Selected papers from the symposium held at Reykjavík University. IEEE Press, Los Alamitos, CA, 2017, 9 pp. [MR 3776918](#) [IEEEExplore 8005092](#) q.v. [275](#)
- [26] B. Khoussainov and A. Nerode, *Automata theory and its applications*. Progress in Computer Science and Applied Logic, 21. Birkhäuser Boston, Boston, MA, 2001. [MR 1839464](#) [Zbl 1083.68058](#) q.v. [266](#)
- [27] N. Klarlund, Progress measures, immediate determinacy, and a subset construction for tree automata. *Ann. Pure Appl. Logic* 69 (1994), no. 2–3, 243–268. Invited papers presented at the 1992 IEEE Symposium on Logic in Computer Science. Held in Santa Cruz, CA, June 22–25, 1992. [MR 1300675](#) [Zbl 0821.68084](#) [IEEEExplore 185550](#) q.v. [276](#)
- [28] O. Kupferman, S. Safra, and M. Y. Vardi, Relating word and tree automata. In *Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science*. New Brunswick, N.J., July 27–30, 1996. IEEE Press, Los Alamitos, CA, 1996, 322–332. [MR 1461845](#) [IEEEExplore 561360](#) q.v. [269](#)
- [29] O. Kupferman and M. Y. Vardi, Safraless decision procedures. In *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS ’05)*. Held in Pittsburgh, PA, October 23–25, 2005. IEEE Press, Los Alamitos, CA, 2005, 531–542. [IEEEExplore 1530745](#) q.v. [293](#)
- [30] G. Lenzi, A hierarchy theorem for the mu-calculus. In *Automata, languages and programming* (F. M. auf der Heide and B. Monien eds.) Proceedings of the 23rd International Colloquium (ICALP ’96) held at Paderborn University, Paderborn, July 8–12, 1996. Lecture Notes in Computer Science, 1099. Springer, Berlin, 1996, 87–97. [MR 1464442](#) [Zbl 1045.03516](#) q.v. [288](#)
- [31] C. Löding, *Automata and logics over infinite trees*. Habilitationsschrift, RWTH Aachen, Aachen, 2009. q.v. [285](#)

- [32] S. Miyano and T. Hayashi, Alternating finite automata on ω -words. *Theoret. Comput. Sci.* 32 (1984), no. 3, 321–330. [MR 0761350](#) [Zbl 0544.68042](#) q.v. 279
- [33] A. W. Mostowski, Determinacy of sinking automata on infinite trees and inequalities between various pairs of Rabin's indices. *Inform. Process. Lett.* 15 (1992), no 4, 159–163. [MR 0582.68023](#) q.v. 283, 284
- [34] A. W. Mostowski, Regular expressions for infinite trees and a standard form of automata. In *Computation theory* (A. Skowron, ed.). Proceedings of the 5th symposium held in Zaborów, December 3–8, 1984. Lecture Notes in Computer Science, 208. Springer, Berlin, 1985, 157–168. [MR 0827531](#) [Zbl 0612.68046](#) q.v. 268
- [35] A. W. Mostowski, Games with forbidden positions. Technical Report 78, Uniwersytet Gdański, Instytut Matematyki, Gdański, 1991. q.v. 275
- [36] D. E. Muller, Infinite sequences and finite machines. In *Proceedings of the 4th Annual Symposium on Switching Circuit Theory and Logical Design*. SWCT 1963. Held in Chicago, IL, October 28–30, 1963. IEEE Press, Los Alamitos, CA, 1963, 3–16. [IEEEExplore 4569783](#) q.v. 267
- [37] D. E. Muller, A. Saoudi, and P. E. Schupp, Alternating automata, the weak monadic theory of trees and its complexity. *Theoret. Comput. Sci.* 97 (1992), no. 2, 233–244. [MR 1163817](#) [Zbl 0776.03017](#) q.v. 273, 290, 291
- [38] D. E. Muller and P. E. Schupp, Alternating automata on infinite trees. *Theoret. Comput. Sci.* 54 (1987), no. 2–3, 267–276. [MR 0919595](#) [Zbl 0636.68108](#) q.v. 276, 277, 279
- [39] D. E. Muller and P. E. Schupp, Simulating alternating tree automata by nondeterministic automata: New results and new proofs of the theorems of Rabin, McNaughton and Safra. *Theoret. Comput. Sci.* 141 (1995), no. 1-2, 69–107. [MR 1323149](#) [Zbl 0873.68135](#) q.v. 282, 283
- [40] D. Niwiński, On fixed-point clones (extended abstract). In *Automata, languages and programming* (L. Kott, ed.). Proceedings of the 13th International Colloquium (ICALP '86) held in Rennes, July 15–19, 1986. Lecture Notes in Computer Science, 226. Springer, Berlin, 1986, 464–473. [MR 0864709](#) [Zbl 0596.68036](#) q.v. 288
- [41] D. Niwiński and I. Walukiewicz, Relating hierarchies of word and tree automata. In *STACS '98* (M. Morvan, C. Meinel, and D. Krob, eds.). Proceedings of the 15th Annual Symposium on Theoretical Aspects of Computer Science held in Paris, February 25–27, 1998. Lecture Notes in Computer Science, 1373. Springer, Berlin, 1998, 320–331. [MR 1650682](#) [Zbl 0892.68067](#) q.v. 269, 288
- [42] D. Niwiński and I. Walukiewicz, Deciding nondeterministic hierarchy of deterministic tree automata. In *Proceedings of the 11th Workshop on Logic, Language, Information and Computation* (R. de Queiroz and P. Cégielski, eds.). WoLLIC 2004. Held at the Université Paris 12, Fontainebleau, July 19–22, 2004. Elsevier, Amsterdam, 2005, 195–208. [MR 2207417](#) q.v. 288
- [43] D. Perrin and J.-É. Pin, *Infinite words*. Automata, semigroups, logic and games. Pure and Applied Mathematics (Amsterdam), 141. Elsevier/Academic Press, 2004. [Zbl 1094.68052](#) q.v. 266
- [44] A. Pnueli and R. Rosner, On the synthesis of a reactive module. In *Proceedings of the 16th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*. POPL '89. Austin, TX, January 11–13, 1989. Association for Computing Machinery, New York, 1989, 179–190. q.v. 293

- [45] M. O. Rabin, Decidability of second-order theories and automata on infinite trees. *Trans. Amer. Math. Soc.* 141 (1969), 1–35. [MR 0246760](#) [Zbl 0221.02031](#)
q.v. [265](#), [268](#), [282](#), [288](#), [289](#), [292](#)
- [46] M. O. Rabin, Weakly definable relations and special automata. In *Mathematical logic and foundations of set theory* (Y. BarHillel, ed.). Proceedings of an International Colloquium held under the auspices of the Israel Academy of Sciences and Humanities, Jerusalem, November 11–14, 1968. Studies in Logic and the Foundations of Mathematics. North-Holland Publishing Co., Amsterdam and London, 1970, 1–23. [MR 0277388](#) [Zbl 0214.02208](#)
q.v. [290](#), [291](#)
- [47] M. O. Rabin, Automata on infinite objects and Church’s problem. Conference Board of the Mathematical Sciences Regional Conference Series in Mathematics, 13. American Mathematical Society, Providence, R.I., 1972. [MR 0321708](#) [Zbl 0315.02037](#)
q.v. [269](#), [286](#), [287](#), [292](#), [293](#)
- [48] S. Safra, On the complexity of ω -automata. In *29th Annual Symposium on Foundations of Computer Science*. Held in White Plains, N.Y., October 24–26, 1988. IEEE Press, Los Alamitos, CA, 1988, 319–327. [IEEEExplore 21948](#) q.v. [269](#), [282](#)
- [49] S. Safra, Exponential determinization for ω -automata with a strong fairness acceptance condition. *SIAM J. Comput.* 36 (2006), no. 3, 803–814. [MR 2263013](#) [Zbl 1120.68072](#)
q.v. [283](#)
- [50] S. Schewe, Solving parity games in big steps. *J. Comput. System Sci.* 84 (2017), 243–262. [MR 3570179](#) [Zbl 1353.68180](#) q.v. [275](#)
- [51] H. Seidl, Deciding equivalence of finite tree automata. *SIAM J. Comput.* 19 (1990), no. 3, 424–437. [MR 1041537](#) [Zbl 0699.68075](#) q.v. [284](#), [287](#)
- [52] R. E. Stearns and H. B. Hunt III, On the equivalence and containment problems for unambiguous regular expressions, regular grammars and finite automata. *SIAM J. Comput.* 14 (1985), no. 3, 598–611. [MR 0795933](#) [Zbl 0577.68074](#) q.v. [284](#)
- [53] L. J. Stockmeyer, The complexity of decision problems in automata theory and logic. Ph.D. thesis, Department of Electrical Engineering, MIT, Cambridge, MA, 1974. q.v. [292](#)
- [54] R. S. Streett, Propositional dynamic logic of looping and converse is elementarily decidable. *Inform. and Control* 54 (1982), no. 1–2, 121–141. [MR 0713309](#) [Zbl 0515.68062](#) q.v. [268](#)
- [55] W. Thomas, Automata on infinite objects. In *Handbook of theoretical computer science* (J. van Leeuwen, ed.). Vol. B. Formal models and semantics. Elsevier Science Publishers, Amsterdam, and MIT Press, Cambridge, MA, 1990, 133–191. [MR 1127189](#)
[Zbl 0900.68316](#) q.v. [265](#), [294](#)
- [56] W. Thomas, Languages, automata, and logic. In *Handbook of formal languages* (G. Rozenberg and A. Salomaa, eds.). Vol. 3. Beyond words. Springer, Berlin, 1997, 389–455. [MR 1470024](#) q.v. [265](#), [269](#), [273](#), [275](#), [280](#), [283](#), [286](#), [290](#)
- [57] M. Y. Vardi, Reasoning about the past with two-way automata. In *Automata, languages and programming* (K. G. Larsen, S. Skyum, and G. Winskel, eds.). Proceedings of the 25th International Colloquium (ICALP ’98) held in Aalborg, July 13–17, 1998. Lecture Notes in Computer Science, 1443. Springer, Berlin, 1998, 628–641. [MR 1683554](#) [Zbl 0909.03019](#)
q.v. [283](#)

- [58] M. Y. Vardi and T. Wilke, Automata: from logics to algorithms. In *Logic and automata* (J. Flum, E. Grädel, and T. Wilke, eds.). History and perspectives. Texts in Logic and Games, 2. Amsterdam University Press, Amsterdam, 2008, 629–724. [MR 2508757](#) [Zbl 1234.03026](#) q.v. [266](#), [269](#), [282](#), [294](#)
- [59] J. Virágh, Deterministic ascending tree automata I. *Acta Cybernet.* 5 (1980), no. 1, 33–42. [MR 0599042](#) [Zbl 0454.68044](#) q.v. [284](#)
- [60] I. Walukiewicz, Pushdown processes: games and model-checking. *Inform. and Comput.* 164 (2001), no. 2, 234–263. [MR 1816150](#) [Zbl 1003.68072](#) q.v. [275](#)
- [61] I. Walukiewicz, Deciding low levels of tree-automata hierarchy. In *Proceedings of the 9th workshop on logic, language, information and computation*. WoLLIC 2002. Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio), Rio de Janeiro, Brazil, July 30–August 2, 2002. Electronic Notes in Theoretical Computer Science, 67. Elsevier, Amsterdam, 61–75. [Zbl 1261.68082](#) q.v. [288](#)
- [62] W. Zielonka, Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theoret. Comput. Sci.* 200 (1998), no. 1–2, 135–183. [MR 1625527](#) [Zbl 0915.68120](#) q.v. [266](#), [275](#), [276](#), [286](#)