Automata on Inifinite Trees

Christof Löding

Lehrstuhl Informatik 7
RWTH Aachen
52056 Aachen, Germany
email: loeding@cs.rwth-aachen.de

Draft, December 8, 2011

2010 Mathematics Subject Classification:

Key words:

Abstract. This article presents the basic theory of automata on infinite trees. We introduce the standard model of nondeterministic automata and the generalization to alternating automata. The central part of the theory considers the closure properties of tree languages definable by such automata, the equivalence of the different models, and their algorithmic properties. We show how these results are obtained and how games of infinite duration can be used as a powerful tool to obtain transparent proofs and algorithms. We also present some applications of tree automata in logic, in particular decidability results and characterizations for monadic second-order logic over infinite trees, and algorithms for the modal μ-calculus.

Contents

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 behaviors 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 behavior 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 behaviors 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 has first been investigated in [39] 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 [39] states that a set of infinite labeled trees can be defined in monadic second-order logic if, and only if, it can be recognized 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 [47], which gives an overview of the most fundamental results for automata on infinite words and infinite trees, and the later survey [48] that also includes automata and logic on finite words and trees. The article of Zielonka [53] 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 [22] and [37].

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 [50] 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 Section 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. Section 3 contains central constructions and proofs, namely the connection of tree automata and infinite games (Section 3.1), the complementation theorem for alternating automata (Section 3.2), and the simulation theorem for the transformation of alternating into nondeterministic automata (Section 3.3). Based on these constructions we give some results on the complexity of decision problems for automata on infinite trees in Section 4, and applications in logic in Section 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 and are briefly discussed in Section 5.2 on branching time logics.

Let $k \in \mathbb{N}$ be a natural number and denote by $[k]$ the set $\{0, \ldots, k-1\}$. 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 $i \in [k]$ 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 of u.[1]

A *path* π is an infinite sequence $u_0, u_1, u_2 \ldots$ of successive nodes starting in the root, i.e., $u_0 = \varepsilon$ and for each $j \geqslant 1$ 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-labeled k-ary *tree* is a mapping $t : [k]^* \to A$. 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-labeled and/or k-ary.

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

2.1 Nondeterministic automata

A nondeterministic tree automaton on A-labeled 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 $t \in \mathcal{T}^\omega_{A,k}$ is defined in terms of its transition structure: It is a tree $\rho \in \mathcal{T}^\omega_Q$ such that for each $u \in [k]^*$ we have $(\rho(u), t(u), \rho(u.0), \ldots, \rho(u.k-1)) \in \Delta$. 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

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

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 $Acc \subseteq Q^\omega$ of infinite sequences of states. A run ρ is accepting if each path satisfies the acceptance condition, i.e., $\rho(\pi) \in Acc$ 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 condition and the way they are specified:

- A *Büchi condition* [3] 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: $Inf(\pi) \cap F \neq \emptyset$.
- 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: $Inf(\pi) \cap F = \emptyset$.
- A *Muller condition* [30] 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: $Inf(\pi) \in \mathcal{F}$.
- A *Rabin condition* [39] is given by a list $\langle (E_1, F_1), \ldots, (E_k, F_k) \rangle$ of pairs of state sets. An infinite sequence π of states satisfies the Rabin condition if for some i it contains only finitely many states from E_i and infinitely many states from F_i: $Inf(\pi) \cap E_i = \emptyset$ and $Inf(\pi) \cap F_i \neq \emptyset$ for some $i \in \{1, \ldots, k\}$.
- A *Streett condition* [46] is given by a list $\langle (E_1, F_1), \ldots, (E_k, F_k) \rangle$ of pairs of state sets. An infinite sequence π of states satisfies the Streett condition if for all i it contains infinitely many states from E_i or only finitely many states from F_i: $Inf(\pi) \cap E_i = \emptyset \rightarrow Inf(\pi) \cap F_i = \emptyset$ for all $i \in \{1, \ldots, k\}$.
- A *parity condition* [28] 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: $\max(Inf(c(\pi)))$ 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: $T(\mathcal{A}) = \{t \in \mathcal{T}^\omega_{A,k} \mid t \text{ accepted by } \mathcal{A}\}$.

Example 2.1. Let $A = \{a, b\}$ and $T_{\exists b} \subseteq \mathcal{T}^\omega_{A,k}$ 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 labeled q_b to the left or to the right. If it finds some b on this path, it can switch 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 [39].

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 $Acc \subseteq Q^\omega$ 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.

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 induces 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. □

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 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 b (see e.g. [48] for a proof).

Proposition 2.2 ([41]). *The language of all trees over $\{a, b\}$ that have only finitely many b on each path cannot be recognized by a Büchi tree automaton.*

The relation between recognizability of a word language and the induced tree language can be generalized as follows. For a language L of infinite words let $Path(L)$ be the language of all trees such that each path is labeled by a word in L. In [24] it is shown that a language of the form $Path(L)$ can be accepted by a nondeterministic Büchi tree

automaton if, and only if, L can be accepted by a deterministic Büchi word automaton. In [35] this result is extended to parity automata:

Theorem 2.3 ([35]). *A tree language of the form $Path(L)$ can be accepted by a nondeterministic parity tree automaton using priorities $i, \ldots, j$ if, and only if, L can be accepted by a deterministic parity word automaton using priorities $i, \ldots, j$.*

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 Section 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 generalize 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 $\wedge$, $\vee$). The empty conjunction is always true and is denoted by $\mathbf{tt}$, while the empty disjunction is always false and is denoted by $\mathbf{ff}$. For $\varphi \in \mathcal{B}^+(X)$ and $Y \subseteq X$ we write $Y \models \varphi$ if the truth assignment that assigns true to all elements from Y and false to all elements of $X \setminus Y$ makes φ true.

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 $\delta : Q \times A \to \mathcal{B}^+(Q \times [k])$. 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 $\delta(q, a) = \bigvee_{(q,a,q_0,\ldots,q_{k-1})\in\Delta}(q_0, 0) \wedge \cdots \wedge (q_{k-1}, k-1)$. For nondeterministic automata we refer to the elements $(q, a, q_1, \ldots, q_k)$ 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, $T = \{t \in \mathcal{T}_A^\omega \mid \forall u \in [2]^*(t(u) = a \to \exists v \in [2]^*(t(uv) = b))\}$.

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 looks as follows:

$$\begin{array}{lcl} \delta(q, a) & = & (q, 0) \wedge (q, 1) \wedge \big((q_b, 0) \vee (q_b, 1)\big) \\ \delta(q, b) & = & (q, 0) \wedge (q, 1) \\ \delta(q_b, a) & = & (q_b, 0) \vee (q_b, 1) \\ \delta(q_b, b) & = & \mathbf{tt} \end{array}$$

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 labeled 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 for nondeterministic automata. For this reason, a run R is a forest with vertices labeled by states and by nodes

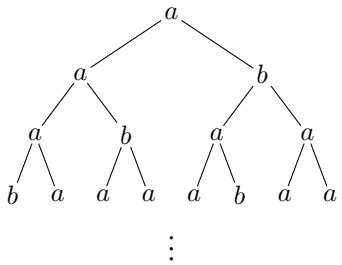

Figure 1. A labeled binary tree.

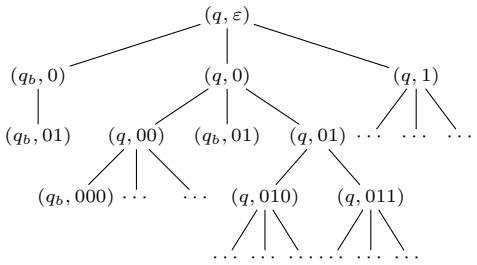

Figure 2. A run of the automaton from Example 2.2 on the tree from Figure 1.

of the tree. The paths through the forest correspond to computations of the automaton along paths of the input tree.

Let $\mathcal{A}$ be a alternating tree automaton with transition structure (Q, A, I, δ), and let $t \in \mathcal{T}^{\omega}_{A,k}$ 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 $\rho : V_R \to Q \times [k]^*$ 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 labeled by a pair (q, ε), and the set $\{q_1, \ldots, q_n\}$ of state labels of the roots of R satisfies the initial formula I.
- Let $v \in V_R$ be a node of v with $\rho(v) = (q, u)$, and let $v_1, \ldots, v_n$ 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 $(q, t(u), \{(q_i, d_i) \mid 1 \leqslant i \leqslant n\})$ 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.

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 denote the set of all trees for which there is an accepting run of $\mathcal{A}$ by $T(\mathcal{A})$. 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 [31]. 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 formalized by using an ordered partition of the state space, i.e., Q is the disjoint union of sets $Q_1, \ldots, Q_m$ 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 $i \leqslant j$. Furthermore, each Q_i is either classified as accepting or as rejecting. Note that the labels on each path through a run of a weak automaton eventually stabilize to 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 using the partition $Q_1 = \{q\}$ and $Q_2 = \{q_b\}$, and classifying Q_1 as accepting and Q_2 as 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 characterize a variant of monadic second-order logic. This relation is explained in more detail in Section 5.1.

As for nondeterministic automata we state some simple properties of alternating automata. The more involved constructions are presented in Section 3. The possibility of using disjunctions and conjunctions in alternating automata allows simple constructions for the closure under union and intersection by taking the disjoint union of the two given automata and combining 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 a simple adaption of the pairs is necessary, left as an exercise to the reader).

In Remark 2.5 we have shown that applying a projection to the transition labels of a given nondeterministic automaton yields an automaton for the projected language. This simple construction does not work for alternating automata. For constructing an automaton accepting the projection of the tree language defined by an alternating automaton one first has to transform it into a nondeterministic one. This is the subject of Section 3.3.

3 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 [48].

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 Section 3.1. In Section 3.2 we show how to complement alternating automata, and in Section 3.3 we transform alternating automata into nondeterministic ones.

3.1 Infinite games

The games we are interested in are played on game graphs, also called arenas. An *arena* is a tuple $G = (V_{\mathsf{E}}, V_{\mathsf{A}}, E, c)$, 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 $V = V_{\mathsf{E}} \cup V_{\mathsf{A}}$ is the set of edges or moves, and $c : V \to C$ is a mapping that assigns to each vertex a color from a finite set C of colors.

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 vertcies 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 colors. The winning condition specifies whether this sequence is winning for Eva or for Adam. Hence, formally a *winning condition* is given by the set $Win \subseteq C^\omega$ of infinite plays that are winning for Eva. A *game* is a pair $\mathcal{G} = (G, Win)$ 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 $\sigma_{\mathsf{E}} : V^*V_{\mathsf{E}} \to V$ that takes a finite prefix of a play ending in a vertex of Eva, and outputs her next move, with the property that $\sigma_{\mathsf{E}}(wv) = v'$ 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 $\sigma_{\mathsf{A}} : V^*V_{\mathsf{A}} \to V$ such that $\sigma_{\mathsf{A}}(wv) = v'$ 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 $\gamma(i) \in V_{\mathsf{E}}$ the next vertex, if it exsists, is obtained by applying the strategy: $\sigma_{\mathsf{E}}(\gamma[0, i]) = \gamma(i + 1)$. For a strategy of Adam the definition is analogous.

Given a game $\mathcal{G} = (G, Win)$ and an initial vertex v_0, a strategy σ_{E} for Eva is a *winning strategy* from v_0 if all possible 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 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 Section 2, that is, the coloring is of the form $c : V \to \mathbb{N}$. To determine the winner of a play γ we look at the maximal number that appears infinitely often in $c(\gamma)$. If this color is even, then Eva wins, otherwise Adam wins.

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 coloring function c. Hence, parity games are completely specified by the game arena $(V_{\mathsf{E}}, V_{\mathsf{A}}, E, c)$.

Parity games are important because they are not only determined but the determinacy result already holds when restricting to a very simple type of 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 $\sigma_{\mathsf{E}} : V_{\mathsf{E}} \to V$ such that $(v, \sigma_{\mathsf{E}}(v)) \in E$ 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 ([13, 29]). *Parity games are determined with uniform positional strategies for both players on their respective winning areas.*

There are various proofs of this result. The proofs in the survey articles [53, 48] are based on an induction on the number of priorities. On finite graphs this kind of inductive proof yields an algorithm for solving parity games roughly in time $\mathcal{O}(n^d)$ where n is the number of vertices and d the number of priorities. In [51] Walukiewicz gives a proof based on the concept of signatures or progress measures. On finite graphs this concept is used to derive an algorithm for computing the winning regions in [20].

There are many other algorithms for solving finite parity games (see Chapter 7 of [17] for an overview, and the more recent works [21] and [44]). The most obvious algorithm can directly be derived from the positional determinacy: On a finite graph one can guess a positional strategy and verify it 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 NP $\cap$ co-NP. We summarize the above by the following theorem.

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

The question whether parity games can be solved in polynomial time is one of the main open questions in this area.

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

Theorem 3.3 ([23]). *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 general Muller games both players might need memory. One can, however, show that finite memory only depending on the number of colors is sufficient for winning strategies, which then can be implemented by finite automata (see [17, 53] for an overview of the main results).

Our next goal is to characterize the semantics of a an alternating tree automaton $\mathcal{A}$ in terms of a game, which has first been described in [18] for nondeterministic tree automata, and later been adapted for alternating tree automata [32]. 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 consist only of two rounds but the players choose very complex objects. We now 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 (q', d) with $d \in [k]$. The game continues in $(q', x.d)$.

For simplicity, we consider automata with a single initial state, which is sufficient as mentioned in Section 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 from $Q \times [k]^*$ belong to Eva.
- 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 $\psi = \mathbf{tt}$. Otherwise they belong to Eva (disjunctions, $\mathbf{ff}$, and atoms). From the definition of the edge relation below it follows that atoms 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 $\psi = \psi_1 \; op \; \psi_2$ with $op \in \{\wedge, \vee\}$. Then we have the following edges in $E_{\mathcal{A},t}$:

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

- The set of colors is the set Q of states of $\mathcal{A}$ together with a color $\perp$. To each node of the form (q, x) we assign the color q, and to all other nodes the color $\perp$. 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 lines the sub-game at the corresponding node is shown: A vertex (q, x) of the game is displayed as q inside the dashed line 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

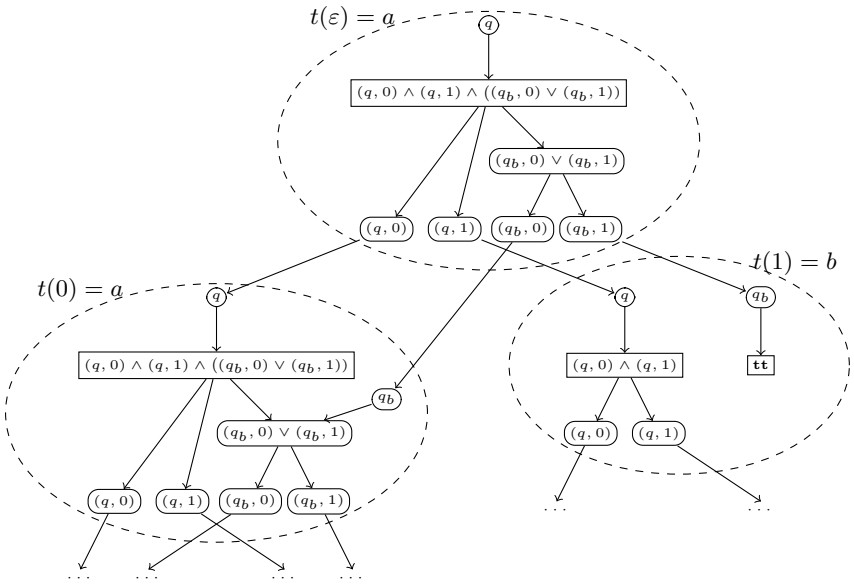

Figure 3. Initial reachable part of the membership game for the automaton from Example 2.2 and the tree from Figure 1.

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 ([18, 32]). *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: (q, a, P) is a transition of $\mathcal{A}$ if, and only if, Eva has a strategy in $\mathcal{G}_{\mathcal{A},t}$ to ensure 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 $\delta(q, a)$.

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. □

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 as alternating ones, then the transition formulas are of a very restricted shape, they are disjunctions of conjunctions, where each conjunctions 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 $\widetilde{\mathcal{A}}$ 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 $\widetilde{\mathcal{A}}$ 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 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 dualize $\mathcal{G}_{\mathcal{A},t}$ as explained above. We call this formula obtained from φ by exchanging $\wedge$ with $\vee$ and $\mathbf{tt}$ with $\mathbf{ff}$ the dual formula of φ and denote it by $\widetilde{\varphi}$.

For an alternating tree automaton $\mathcal{A}$ with transition function δ we define the dual transition function $\widetilde{\delta}$ by $\widetilde{\delta}(q,a) = \widetilde{\delta(q,a)}$ 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 $(Q, A, \widetilde{I}, \widetilde{\delta})$. The dual $\widetilde{A}$ of an alternating tree automaton $\mathcal{A}$ is obtained by dualizing its transition structure and its acceptance condition. An abstract acceptance condition Acc is dualized by simply taking its complement $Q^\omega \setminus Acc$. When working with concrete acceptance conditions, then dualization 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 $\langle (E_1, F_1), \ldots, (E_k, F_k) \rangle$ 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), and for weak automata dualization can be done by flipping the status of the components between accepting and non-accepting.

Since dualization 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 $\mathcal{G}_{\widetilde{\mathcal{A}},t}$.*

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

Theorem 3.7 ([32]). *Let $\mathcal{A}$ be an alternating tree automaton and $\widetilde{\mathcal{A}}$ be its dual. Then $\mathcal{T}^\omega_{A,k} \setminus T(\mathcal{A}) = T(\widetilde{\mathcal{A}})$.*

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

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

of the subset construction. Assume that we take a run of the given alternating automaton and we simply merge all vertices of the run that are associated to the same node in the tree. We obtain a set of states. The nondeterministic automaton computes such sets of states with some additional information for keeping track of the acceptance condition on individual paths.

Theorem 3.8 ([27]). *For each alternating Büchi tree automaton with n states one can construct an equivalent nondeterministic Büchi tree automaton with at most 3^n states.*

We now turn to the full simulation theorem. The strategy we follow here corresponds to the one for complementing nondeterministic parity tree automata as presented in [18] and [48]. It relies on positional determinacy of parity games and determinization 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 Section 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 $\hat{\sigma}_x : \mathcal{B}^+(Q \times [k]) \to \mathcal{B}^+(Q \times [k])$ 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 denote the set of all local strategies by Σ. 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 membership strategy for Eva is a tree $s \in \mathcal{T}^{\omega}_{A \times \Sigma, k}$. 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 is that we can construct a top-down deterministic parity tree automaton accepting precisely the representations of winning strategies of Eva.

Lemma 3.9. *There is a top-down deterministic 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 $\mathcal{T}^{\omega}_{A,k}$ obtained by projection to the A-component, and let σ be the strategy of Eva in $\mathcal{G}_{\mathcal{A},t}$ 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 $atoms(\hat{\sigma},\psi)$ of atoms that a local strategy $\hat{\sigma}$ induces, starting from some element $\psi \in \mathcal{B}^+(Q\times[k])$:

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

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

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 $q_0q_1\cdots$ 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 $q_0q_1\cdots$ corresponds to the state sequence of a play according to σ along the path $i_0i_1\cdots$. 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 $q_0q_1\cdots$ that starts in the initial state and 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 determinizing and complementing this Büchi automaton. From this word automaton one easily constructs a top-down deterministic 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: $(q,(a,\hat{\sigma}),q_1,\ldots,q_k)$ 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 nondeterministic 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 determinization construction. The Büchi automaton guesses a sequence

$q_0 q_1 \cdots$ 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 $q_0 q_1 \cdots$ 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 $\mathcal{A}$. Determinization yields a deterministic parity word automaton with at most $2^{\mathcal{O}(nd \log(nd))}$ states and $\mathcal{O}(nd)$ priorities [42]. □

The simulation result is a direct consequence of Lemma 3.9.

Theorem 3.10 ([33]). *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 Adam. Omitting the Σ-component from the labels in the transitions yields a (nondeterministic) parity tree automaton that accepts all trees $t \in \mathcal{T}^{\omega}_{A,k}$ on which Eva has a winning strategy in $\mathcal{G}_{\mathcal{A},t}$. According to Lemma 3.5 this is exactly the language of $\mathcal{A}$. □

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

Corollary 3.11 ([39]). *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 determinization 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 the next section. We conclude this section with some remarks on the history and further results on constructions for automata over infinite trees.

The complementation result was already shown by Rabin in [39] also relying on McNaughton's theorem on determinization 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 proven by Muller and Schupp in [33] for alternating Streett tree automata. Their proof 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 with 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 [33] yields a nondeterministic automaton with $2^{\mathcal{O}nd\log(n)}$ 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 [18] and is very nicely presented in [48].

We have presented the simulation construction for parity tree automata. The same techniques also work for Rabin conditions with the same complexities: 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 using a determinization result for nondeterministic Streett automata [43].

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 [49].

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, 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 Section 5). Furthermore, other problems like 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 Section 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, one has to be careful because 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 $(q, a, q_0, \ldots, q_{k-1})$ with q as source state.
- From a transition $(q, a, q_0, \ldots, q_{k-1})$ Adam can move to one of the states $q_0, \ldots, q_{k-1}$. This corresponds to the choice of a direction $i \in [k]$.

As priority function we simply take the priority function of $\mathcal{A}$ and extend it to transitions by setting $c(q, a, q_0, q_1) = c(q)$. We can now relate the existence of winning strategies in $\mathcal{G}_\mathcal{A}$ to the emptiness of $T(\mathcal{A})$ [53, 48]:

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

The size of the parity game $\mathcal{G}_\mathcal{A}$ is linear in the size of $\mathcal{A}$. 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 [41].

Theorem 4.2. *The emptiness problem for parity tree automata is in NP ∩ co-NP. If the automaton works over k-ary trees, has n states, m transitions, and d priorities, then the emptiness problem can be solved in time $\mathcal{O}(km(n+m)^d)$.*

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 $\mathcal{B}$, we can test $T(\mathcal{A}) \subseteq T(\mathcal{B})$ by combining $\mathcal{A}$ and $\widetilde{\mathcal{B}}$ into a parity tree automaton accepting $T(\mathcal{A}) \cap (\mathcal{T}^\omega_{A,k} \setminus T(\mathcal{B}))$ 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. □

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 [45], 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 $t \in \mathcal{T}^{\omega}_{A,k}$ 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 $\lambda : Q \to A$, 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 labeled 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 $\mathcal{G}_{\mathcal{A}}$ we can construct a regular tree in the language of $\mathcal{A}$.

Theorem 4.4 ([41]). *Every non-empty regular tree language contains a regular tree. The number of states of the automaton generating the tree is the same as the number of states of $\mathcal{A}$.*

Proof. A positional strategy σ_{E} for Eva chooses a transition $(q, a, q_0, \ldots, q_{k-1})$ 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 $i \in [k]$. □

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 using automata. 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 Section 3.1 we have introduced the membership game $\mathcal{G}_{\mathcal{A},t}$ characterizing the membership of t in the language $T(\mathcal{A})$. For general trees t the arena of $\mathcal{G}_{\mathcal{A},t}$ is infinite but it is easy to see that the arena can be made finite if the tree t is regular, by identifying nodes of the tree that are roots of isomorphic subtrees (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 $\mathcal{O}((|\mathcal{A}| \cdot |t|)^d)$.*

We finish this section by briefly discussing the parity index problem for regular tree languages: Given a regular tree language T and a range $i, \ldots, j$ of priorities, decide if T can be accepted by a parity automaton using the interval $i, \ldots, j$ 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 ([34] and [2, 26]). 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, that is, deciding for a given tree language and priority range $i, \ldots, j$ whether T can be accepted by a nondeterministic parity tree automaton with priorities $i, \ldots, j$.

If the given language is of the form $Path(L)$ for an ω-language L (see Section 2.1), then we can use Theorem 2.3 and the fact that for deterministic parity word automata the index problem is decidable [35, 8]. This decidability result can be generalized to the case where the given tree language can be accepted by a deterministic tree automaton [36]. The lowest level of the hierarchy, the case where $i = j = 0$ is known to be decidable for arbitrary regular tree languages as input [52]. An approach to tackle the general problem has been presented in [10] 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 [39]. In Section 5.1 we present Rabin's result on the equivalence of MSO and tree automata, and also give a characterization 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 synthesizing 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 Section 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. [12] 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. First order variables are usually denoted by small letters (e.g. x, y), and monadic second-order variables are denoted by capital letters (e.g. X, Y). First-order logic (FO) is the fragment of MSO that does not use set quantifications.

We view the unlabeled k-ary tree as the relational structure $\mathfrak{t}_2 = ([k]^*, E_0, \ldots, E_k)$, where the E_i are binary symbols interpreted as $\{(w, w.i) \mid w \in [k]^*\}$. MSO logic over $\mathfrak{t}_2$ is also referred to as S2S (second order theory of two successors).

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

The tree language $T(\varphi) \subseteq \mathcal{T}^{\omega}_{[2]^n}$ of φ is $T(\varphi) = \{t[U_1, \ldots, U_n] \mid \mathfrak{t}_2 \models \varphi[U_1, \ldots, U_n]\}$, where $\varphi[U_1, \ldots, U_n]$ means that each variable X_i is interpreted by the set U_i. We also write $t[U_1, \ldots, U_n] \models \varphi$ if $t[U_1, \ldots, U_n] \in T(\varphi)$.

The key result from [39] for showing the decidability of MSO over $\mathfrak{t}_2$ 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 ([39]). *A tree language $T \subseteq \mathcal{T}^{\omega}_{[2]^n}$ 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. [48]).

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 $\mathfrak{t}_2$ the sentence $\forall X \exists x : \ x \notin X$ is wrong as MSO sentence but it is true as WMSO sentence. However, we can translate WMSO formulas into MSO formulas because finiteness of sets can be expressed in MSO over $\mathfrak{t}_2$: A set of nodes of $\mathfrak{t}_2$ is finite if there exists a cut through the tree (a set of nodes pairwise incomparable w.r.t. the prefix relation that intersects every maximal 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 [40] 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 characterization of WMSO definable languages in terms of weak alternating automata has been given [31].

Theorem 5.2 ([40, 31]). *For a language $T \subseteq \mathcal{T}^{\omega}_{[2]^n}$ the following are equivalent:*

[3] Rabin used the notion "special automaton".

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

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 Section 4.

Theorem 5.3 ([39]). *The satisfiability problem for MSO over t_2 is decidable.*

If a formula $\varphi(X_1, \ldots, X_n)$ 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 5.4 ([41]). *Let $\varphi(X_1, \ldots, X_n)$ be a satisfiable MSO formula over t_2. There are regular sets $U_1, \ldots, U_n \subseteq [2]^*$ such that $t[U_1, \ldots, U_n] \models \varphi$.*

Using the decidability theorem for S2S as a starting point, one can obtain large classes of structures with a decidable MSO theory. For example, the Caucal hierarchy [9] is a class of graphs that is obtained from finite graphs by iterated application of unfolding and MSO interpretation.[4]

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.

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. In [11] it is shown that the unfolding operation is preserves decidability of MSO.

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. The unfolding of a graph can be reconstructed from its iteration by an MSO interpretation. Details on the iteration operation and its properties can be found in [1].

The method of interpretation mentioned above can also be used to obtain undecidability results. In [7] 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 and therefore the MSO theory of every extension of t_2 by a well-ordering is undecidable.

[4] In the original formulation inverse rational substitution were used instead of interpretations but the resulting classes are equivalent [6].

A related result is the undefinability of a choice function over the infinite binary tree in MSO due to Gurevich and Shelah [19]: There is no MSO formula $\varphi(x, X)$ such that for each nonempty set U there is exactly one $u \in U$ such that $\mathfrak{t}_2 \models \varphi[u, U]$. A proof of this result using automata theoretic methods has been given in [7]. Based on this result one can show that there are regular languages of infinite trees that are ambiguous, that is, there is no *unambiguous automaton* for this language [5]. A nondeterministic automaton is called unambiguous if it has exactly one accepting run on each accepted input.

We finish this section on MSO with the problem of synthesizing a so called reactive module, which reads a stream of input symbols $a_1 a_2 \cdots$ and synchronously produces a stream of output symbols $b_1 b_2 \cdots$ such that the sequence $(a_1, b_1)(a_2, b_2) \cdots$ satisfies a given specification. The task is to automatically synthesize 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 realization of such a specification L is a function $f : A^+ \to B$ such that each sequence of the form $(a_1, f(a_1))(a_2, f(a_1 a_2))(a_3, f(a_1 a_2 a_3)) \cdots$ is in L. The synthesis problem is to decide for a given specification whether it is realizable and to construct a realizing function if possible.

It is clear that the difficulty of this problem depends on how complex the allowed specifications are. In the formulation we have given above, a specification can be any language of infinite words over $(A \times B)$. 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 $\varphi(X_1, \ldots, X_m, Y_1, \ldots, Y_n)$, where the variables $X_1, \ldots, X_m$ code the input sequence, and $Y_1, \ldots, Y_n$ code the output sequence.

The function f we are looking for is of the form $f : ([2]^m)^+ \to [2]^n$. Such a function can easily be represented by a k-ary $[2]^n$-labeled tree t_f with $k = 2^m$. Each node x of t_f naturally corresponds to an input sequence since elements of $[k]$ in their binary representation are 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 a tree automaton (or an MSO formula over $\mathfrak{t}_2$) that accepts precisely the trees of the form t_f for some f realizing φ [41]. A related approach using games has been used by Büchi and Landweber in [4].

Theorem 5.5 ([4, 41]). *The realizability and 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 metioned 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 realizability 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 [38] 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 accepting the admissible behaviors according to the specification, and then running this deterministic automaton over all the branches of a tree of suitable branching degree, where as above the direction in the tree codes the inputs, and the labels of the nodes the outputs. A regular tree (see Section 4) in the language of the resulting automaton provides a finite state program realizing the specification. Since determinization of ω-automata involves complex constructions, some alternative approaches have been developed that avoid explicit determinization constructions [25, 16].

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 that allow to consider arbitrary Kripke structures as models are based on automata for unordered trees and can be found, for example, in [17] and [50]. The underlying techniques, however, are the same in both settings.

Formulas of the μ-calculus are built from a set $P = \{p_1, \ldots, p_n\}$ of atomic propositions, variables $X, Y, \ldots$ denoted by capital letters, 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 least and a 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:

- $[\![p_i]\!]_t^f = \{u \in [2]^* \mid p_i \in t(u)\}$
- $[\![X]\!]_t^f = f(X)$
- $[\![\neg\varphi]\!]_t^f = [2]^* \setminus [\![\varphi]\!]_t^f$ (and similarly for the other Boolean operations)
- $[\![\Diamond\varphi]\!]_t^f = \{u \in [2]^* \mid u.i \in [\![\varphi]\!]_t^f \text{ for some } i \in [2]\}$
- $[\![\Box\varphi]\!]_t^f = \{u \in [2]^* \mid u.i \in [\![\varphi]\!]_t^f \text{ for all } i \in [2]\}$

- $[\![\mu X.\varphi(X)]\!]_t^f = \bigcap\{U \subseteq [2]^* \mid U = [\![\varphi]\!]_t^{f[X\mapsto U]}\}$, where $f[X \mapsto U]$ is the interpretation of variables that interprets X as U and all other variables Y as $f(Y)$.
- $[\![\nu X.\varphi(X)]\!]_t^f = \bigcup\{U \subseteq [2]^* \mid U = [\![\varphi]\!]_t^{f[X\mapsto U]}\}$

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 $\neg\Diamond\varphi \equiv \Box\neg\varphi$ 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 can be directly derived 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 $\Box$ 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 in side 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 fix point formula. However, if we do this infinitely often without quitting the formula $\mu X.\varphi(X)$ (through atomic predicates or 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 of the automaton we have to be careful because, according to the above description, the automaton only moves in the tree for $\Diamond$- and $\Box$-formulas. For this reason, we allow extended transition functions of the form $\delta : Q \times A \to \mathcal{B}^+((Q \times [k]) \cup Q)$, 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 $\mathcal{A}_\psi$ for a μ-calculus sentence ψ (without free variables) has as state set the set of all sub-formulas of ψ. We denote the state for ψ by $\langle\psi\rangle$. The transition function δ is defined as follows:

- $\delta(\langle\lambda X.\varphi\rangle, a) = \langle\varphi\rangle$ for $\lambda \in \{\mu, \nu\}$,
- $\delta(\langle\Diamond\varphi\rangle, a) = (\langle\varphi\rangle, 0) \vee (\langle\varphi\rangle, 1)$,
- $\delta(\langle\Box\varphi\rangle, a) = (\langle\varphi\rangle, 0) \wedge (\langle\varphi\rangle, 1)$,
- $\delta(\langle\psi_1 \wedge \psi_2\rangle, a) = \langle\psi_1\rangle \wedge \langle\psi_2\rangle$,
- $\delta(\langle\psi_1 \vee \psi_2\rangle, a) = \langle\psi_1\rangle \vee \langle\psi_2\rangle$,
- $\delta(\langle p_i\rangle, a) = \mathbf{tt}$ 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.

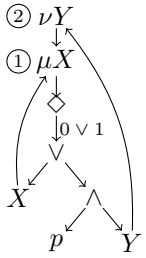

Figure 4. A graphical representation of the automaton for the formula $\nu Y.\mu X.\Diamond((p \wedge Y) \vee X)$

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 to accept it hast to escape 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.

We state the following theorem without proof.

Theorem 5.6 ([14]). *The alternating parity tree automaton $\mathcal{A}_\psi$ accepts precisely those trees t such that ψ holds at the root of t.*

Using the results from Section 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 to decide for a given formula and a regular tree whether the formula holds a the root of the tree.

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 5.7 ([14]). *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 $\mathcal{A}_\psi$. It is not difficult to see 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 5.8 ([15]). *The model-checking problem on regular trees for the μ-calculus can be solved in time $\mathcal{O}((|\psi| \cdot |t|)^d)$ where d is the alternation depth of the formula.*

References

[1] Dietmar Berwanger and Achim Blumensath. The monadic theory of tree-like structures. In E. Grädel, W. Thomas, and T. Wilke, editors, *Automata, Logics, and Infinite Games*, number 2500 in LNCS, chapter 16, pages 285–301. Springer, 2002.

[2] Julian C. Bradfield. The modal mu-calculus alternation hierarchy is strict. *Theoretical Computer Science*, 195(2):133–153, March 1998.

[3] J. Richard Büchi. On a decision method in restricted second order arithmetic. In *International Congress on Logic, Methodology and Philosophy of Science*, pages 1–11. Stanford University Press, 1962.

[4] J. Richard Büchi and Lawrence H. Landweber. Solving sequential conditions by finite-state strategies. *Transactions of the American Mathematical Society*, 138:295–311, 1969.

[5] A. Carayol, C. Löding, D. Niwiński, and I. Walukiewicz. Choice functions and well-orderings over the infinite binary tree. *Central European Journal of Mathematics*, 8(4):662–682, 2010.

[6] Aranud Carayol and Stefan Wöhrle. The caucal hierarchy of infinite graphs in terms of logic and higher-order pushdown automata. In *Proceedings of FST TCS 2003*, volume 2914 of *LNCS*, pages 112–123. Springer, 2003.

[7] Arnaud Carayol and Christof Löding. MSO on the infinite binary tree: Choice and order. In *Proceedings of CSL 2007*, volume 4646 of *LNCS*, pages 161–176. Springer, 2007.

[8] Olivier Carton and Ramón Maceiras. Computing the Rabin index of a parity automaton. *RAIRO Theoretical Informatics and Applications*, 33(6):495–506, 1999.

[9] Didier Caucal. On infinite terms having a decidable monadic theory. In *Proceedings of MFCS 2002*, volume 2420 of *LNCS*, pages 165–176. Springer, 2002.

[10] Thomas Colcombet and Christof Löding. The non-deterministic Mostowski hierarchy and distance-parity automata. In *Proceedings of ICALP 2008*, volume 5126 of *LNCS*, pages 398–409. Springer, 2008.

[11] Bruno Courcelle and Igor Walukiewicz. Monadic second-order logic, graph converings and unfoldings of transition systems. *Annals of Pure and Applied Logic*, 92(1):35–62, 1998.

[12] Heinz-Dieter Ebbinghaus and Jörg Flum. *Finite Model Theory*. Perspectives in Mathematical Logic. Springer, Berlin, 1995.

[13] E. Allen Emerson and Charanjit S. Jutla. The complexity of tree automata and logics of programs (exteded abstract). In *Proceedings of FoCS 1988*, pages 328–337, Los Alamitos, California, October 1988. IEEE Computer Society Press.

[14] E. Allen Emerson and Charanjit S. Jutla. Tree automata, mu-calculus and determinacy (extended abstract). In *Proceedings of FoCS 1991*, pages 368–377, Los Alamitos, California, October 1991. IEEE Computer Society Press.

[15] E. Allen Emerson and Chin-Laung Lei. Efficient model checking in fragments of the propositional mu-calculus (extended abstract). In *Proceedings of LICS 1986*, pages 267–278. IEEE Comuter Society Press, June 1986.

[16] Emmanuel Filiot, Naiyong Jin, and Jean-François Raskin. An antichain algorithm for LTL realizability. In *Proceedings of CAV 2009*, volume 5643 of *LNCS*, pages 263–277. Springer, 2009.

[17] Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors. *Automata, Logics, and Infinite Games*, volume 2500 of *LNCS*. Springer, 2002.

[18] Yuri Gurevich and Leo Harrington. Trees, automata and games. In *Proceedings of STOC 1982*, pages 60–65, 1982.

[19] Yuri Gurevich and Saharon Shelah. Rabin's uniformization problem. *J. Symb. Log.*, 48(4):1105–1119, 1983.

[20] Marcin Jurdziński. Small progress measures for solving parity games. In *Proceedings of STACS 2000*, volume 1770 of *LNCS*, pages 290–301. Springer, February 2000.

[21] Marcin Jurdziński, Mike Paterson, and Uri Zwick. A deterministic subexponential algorithm for solving parity games. In *Proceedings of SODA 2006*, pages 117–123. ACM/SIAM, 2006.

[22] Bakhadyr Khoussainov and Anil Nerode. *Automata theory and its applications*, volume 21 of *Progress in Computer Science and Applied Logic*. Birkhäuser, 2001.

[23] Nils Klarlund. Progress measures, immediate determinacy, and a subset construction for tree automata. *Annals of Pure and Applied Logic*, 69(2–3):243–268, 1994.

[24] Orna Kupferman, Shmuel Safra, and Moshe Y. Vardi. Relating word and tree automata. In *Proceedings of LICS 1996*, pages 322–332. IEEE Computer Society Press, 1996.

[25] Orna Kupferman and Moshe Y. Vardi. Safraless decision procedures. In *Proceedings of FoCS 2005*, pages 531–542. IEEE Computer Society, 2005.

[26] Giacomo Lenzi. A hierarchy theorem for the mu-calculus. In *Proceedings of ICALP 1996*, volume 1099 of *LNCS*, pages 87–97. Springer, July 1996.

[27] Satoru Miyano and Takeshi Hayashi. Alternating finite automata on ω-words. *Theoretical Computer Science*, 32:321–330, 1984.

[28] Andrzej W. Mostowski. Regular expressions for infinite trees and a standard form of automata. In *Computation Theory*, volume 208 of *LNCS*, pages 157–168. Springer, 1984.

[29] Andrzej W. Mostowski. Games with forbidden positions. Technical Report 78, Uniwersytet Gdański, Instytut Matematyki, 1991.

[30] David E. Muller. Infinite sequences and finite machines. In *Proceedings of the 4th IEEE Symposium on Switching Circuit Theory and Logical Design*, pages 3–16, 1963.

[31] David E. Muller, Ahmed Saoudi, and Paul E. Schupp. Alternating automata, the weak monadic theory of the tree, and its complexity. In *Proceedings of ICALP 1986*, volume 226 of *LNCS*, pages 275–283. Springer, 1986.

[32] David E. Muller and Paul E. Schupp. Alternating automata on infinite trees. *Theoretical Computer Science*, 54:267–276, 1987.

[33] David E. Muller and Paul E. Schupp. Simulating alternating tree automata by nondeterministic automata: New results and new proofs of the theorems of Rabin, McNaughton and Safra. *Theoretical Computer Science*, 141(1&2):69–107, 1995.

[34] Damian Niwiński. On fixed-point clones (extended abstract). In *Proceedings of ICALP 1986*, volume 226 of *LNCS*, pages 464–473. Springer, 1986.

[35] Damian Niwiński and Igor Walukiewicz. Relating hierarchies of word and tree automata. In *Proceedings of STACS 1998*, volume 1373 of *LNCS*, pages 320–331. Springer, 1998.

[36] Damian Niwiński and Igor Walukiewicz. Deciding nondeterministic hierarchy of deterministic tree automata. *Electr. Notes Theor. Comput. Sci.*, 123:195–208, 2005.

[37] Dominique Perrin and Jean-Éric Pin. *Infinite words*, volume 141 of *Pure and Applied Mathematics*. Elsevier, 2004.

[38] Amir Pnueli and Roni Rosner. On the synthesis of a reactive module. In *Proceedings of POPL'1989*, pages 179–190, 1989.

[39] Michael O. Rabin. Decidability of second-order theories and automata on infinite trees. *Transactions of the American Mathematical Society*, 141:1–35, July 1969.

[40] Michael O. Rabin. Weakly definable relations and special automata. In Y. BarHillel, editor, *Mathematical Logic and Foundations of Set Theory*, pages 1–23. North-Holland, 1970.

[41] Michael O. Rabin. *Automata on Infinite Objects and Church's Problem*. American Mathematical Society, Boston, MA, USA, 1972.

[42] Shmuel Safra. On the complexity of omega-automata. In *Proceedings of FoCS 1988*, pages 319–327. IEEE Computer Society Press, October 1988.

[43] Shmuel Safra. Exponential determinization for omega-automata with strong-fairness acceptance condition (extended abstract). In *Proceedings of STOC 1992*, pages 275–282, Victoria, British Columbia, Canada, 1992.

[44] Sven Schewe. Solving parity games in big steps. In *Proceedings of FSTTCS 2007*, volume 4855 of *LNCS*, pages 449–460. Springer, 2007.

[45] Helmut Seidl. Deciding equivalence of finite tree automata. *SIAM J. Comput.*, 19(3):424–437, 1990.

[46] Robert S. Streett. Propositional dynamic logic of looping and converse is elementary decidable. *Information and Control*, 54:121–141, 1982.

[47] W. Thomas. Automata on infinite objects. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B: Formal Models and Semantics, pages 133–192. Elsevier Science Publishers, Amsterdam, 1990.

[48] Wolfgang Thomas. Languages, automata, and logic. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Language Theory*, volume III, pages 389–455. Springer, 1997.

[49] Moshe Y. Vardi. Reasoning about the past with two-way automata. In *Proceedings of ICALP'1998*, volume 1443 of *LNCS*, pages 628–641. Springer, 1998.

[50] Moshe Y. Vardi and Thomas Wilke. Automata: from logics to algorithms. In *Logic and Automata - History and Perspectives*, volume 2 of *Texts in Logic and Games*, pages 629–724. Amsterdam University Press, 2007.

[51] Igor Walukiewicz. Pushdown processes: Games and model checking. *Information and Computation*, 164(2):234–263, January 2001.

[52] Igor Walukiewicz. Deciding low levels of tree-automata hierarchy. *Electr. Notes Theor. Comput. Sci.*, 67, 2002.

[53] Wieslaw Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theoretical Computer Science*, 200(1-2):135–183, 1998.