8 Nondeterministic Tree Automata

Frank Nießner

Institut für Informatik
Johann Wolfgang Goethe-Universität Frankfurt am Main

8.1 Introduction

The automaton models introduced so far mainly differ in their acceptance conditions. However, they all consume infinite sequences of alphabet symbols, i.e., they consume ω-words. We therefore call these automata **word automata**. In this chapter we define finite-state automata which process infinite trees instead of infinite words and consequently we call them **tree automata**.

Automata on infinite objects, in general, play an important role in those areas of computer science where nonterminating systems are investigated. System specifications can be translated to automata and thus questions about systems are reduced to decision problems in automata theory. Tree automata are more suitable than words when nondeterminism needs to be modelled.

Furthermore, there are close connections between tree automata and logical theories, which allow to reduce decision problems in logic to decision problems for automata. Such reductions will be thoroughly discussed in Chapter 11. Rabin [148] showed decidability of monadic second-order logic using tree automata which process infinite binary trees. The crucial part in his paper is a complementation theorem for nondeterministic finite-state automata on infinite trees. The proof of this theorem implicitly entails determinacy of parity games. However, Büchi [21] observed that this proof can be much simplified when games are applied explicitly. This approach was successfully implemented by numerous authors, see for instance [77, 55]. Here, we present a game-theoretically based proof of Rabin's theorem according to Thomas [183] and Zielonka [203]. For this purpose we use some results introduced in the previous chapters about infinite games, especially the determinacy theorem for parity games.

Moreover, we consider the emptiness problem for finite-state automata on infinite trees in terms of decidability and efficiency. These observations will be useful in the subsequent chapter about monadic second-order logic.

The chapter is structured as follows. In Section 8.2 we introduce notations and definitions. Section 8.3 introduces two tree automaton models which differ in their acceptance conditions but recognize the same classes of tree languages. We merely sketch the proof of equivalence between the two models. A game-theoretical view on tree automata and their acceptance conditions, together with the main results is given in Section 8.4. Then we are prepared to restate the above-mentioned complementation theorem. The last section, Section 8.5, discusses decidability questions of tree automata. We show that for a particular class of tree automata it is decidable whether their recognized language is empty or not.

E. Grädel et al. (Eds.): Automata, Logics, and Infinite Games, LNCS 2500, pp. 135-152, 2002.
© Springer-Verlag Berlin Heidelberg 2002

8.2 Preliminaries

The **infinite binary tree** is the set $T^\omega = \{0,1\}^*$ of all finite words on $\{0,1\}$. The elements $u \in T^\omega$ are the nodes of T^ω where ε is the root and $u0$, $u1$ are the immediate (say) left and right successors of node u.

We restrict ourselves to binary trees, since they are sufficient for most applications, see, for instance, Chapter 12.

Let $u, v \in T^\omega$, then v is a successor of u, denoted by $u < v$, if there exists a $w \in T^\omega$ such that $v = uw$.

An ω-word $\pi \in \{0,1\}^\omega$ is called a **path** of the binary tree T^ω. The set $Pre_<(\pi) \subset \{0,1\}^*$ of all prefixes of path π (linearly ordered by $<$) describes the set of nodes which occur in π.

For sets Θ, Σ and a mapping $\mu : \Theta \to \Sigma$, we define the **infinity set** $\mathrm{Inf}(\mu) = \{\sigma \in \Sigma \mid \mu^{-1}(\sigma) \text{ is an infinite set}\}$.

We consider here trees where the nodes are labeled with a symbol of an alphabet. A mapping $t : T^\omega \to \Sigma$ labels trees with symbols of Σ. The set of all **Σ-labeled trees** is denoted by T^ω_Σ (or T_Σ for simplicity, if no confusion occurs). Sometimes we are only interested in the **labeling of a path** π through t. Hence let $t|\pi : Pre_<(\pi) \to \Sigma$ denote the restriction of the mapping t to π.

For n an integer and $1 \le i \le n$, the **projection** onto the i-th coordinate is the mapping $p_i : \Sigma^n \to \Sigma$ such that $p_i((\sigma_1, \sigma_2, \ldots, \sigma_n)) = \sigma_i$. We extend projections to labeled infinite trees. For a $\Sigma_1 \times \Sigma_2$-labeled tree $t \in T^\omega_{\Sigma_1 \times \Sigma_2}$, let $p_1(t) \in T^\omega_{\Sigma_1}$ be the corresponding tree labeled exclusively with elements of Σ_1. Projections can be applied to sets as well. Thus a projection $p_1(\Theta)$ of a set $\Theta \subseteq T^\omega_{\Sigma_1 \times \Sigma_2}$ is defined as $p_1(\Theta) = \{p_1(t) \mid t \in \Theta\}$.

Example 8.1. Let $\Sigma = \{a, b\}$, $t(\varepsilon) = a$, $t(w0) = a$ and $t(w1) = b$, $w \in \{0,1\}^*$.

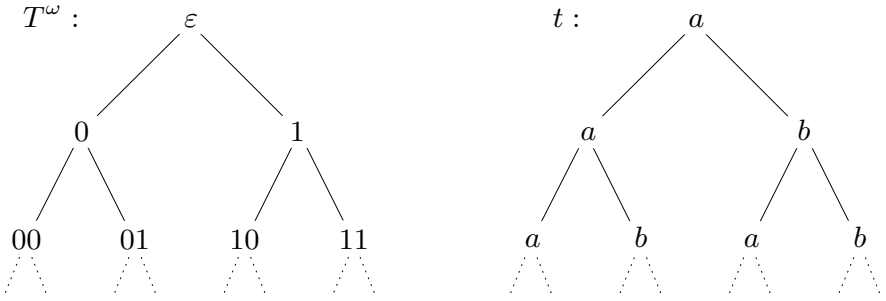

Fig. 8.1. A tree with corresponding labeling

Exercise 8.1. Prove the above-mentioned assertion that binary trees suffice to describe the general case, i.e., describe an encoding of trees with arbitrary finite branching as binary trees.

8.3 Finite-State Tree Automata

The automata seen so far processed finite or infinite sequences of alphabet symbols. They consume one input symbol at a time and thereby enter a successor state determined by a transition relation. It is obvious that we somehow have to modify the automaton models in order to make them running on infinite trees. Since each position in a binary tree has two successors (rather than one successor as in infinite words) it is natural to define for a state out of a set Q and an input symbol from Σ two successor states in the transition relation, that is, transitions are now elements of $Q \times \Sigma \times Q \times Q$. Computations then start at the root of an input tree and work through the input on each path in parallel. A transition (q, a, q_1, q_2) allows to pass from state q at node u with input-tree label a to the states q_1, q_2 at the successor nodes $u0, u1$. Afterwards there may be transitions which allow to continue from q_1 and from q_2, and so on. This procedure yields a Q-labeled tree which we call the **run** of an automaton on an input tree. Such a run is **successful** if all the state sequences along the paths meet an **acceptance condition** similar to the types of acceptance conditions known already from sequential ω-automata.

We start with the description of a **Muller tree automaton**.

Definition 8.2. A Muller tree automaton is a quintuple $\mathcal{A} = (Q, \Sigma, \Delta, q_\mathrm{I}, \mathcal{F})$ where Q is a finite state set, Σ is a finite alphabet, $\Delta \subseteq Q \times \Sigma \times Q \times Q$ denotes the transition relation, q_I is an initial state and $\mathcal{F} \subseteq P(Q)$ is a set of designated state sets. A run of $\mathcal{A}$ on an input tree $t \in T_\Sigma$ is a tree $\varrho \in T_Q$, satisfying $\varrho(\varepsilon) = q_\mathrm{I}$ and for all $w \in \{0,1\}^* : (\varrho(w), t(w), \varrho(w0), \varrho(w1)) \in \Delta$. It is called successful if for each path $\pi \in \{0,1\}^\omega$ the **Muller acceptance condition** is satisfied, that is, if $\mathrm{Inf}(\varrho|\pi) \in \mathcal{F}$. We refer to Section 1.3.2 for a thorough definition of the Muller acceptance condition. $\mathcal{A}$ accepts the tree t if there is a successful run of $\mathcal{A}$ on t. The tree language recognized by $\mathcal{A}$ is the set $T(\mathcal{A}) = \{t \in T_A^\omega \mid \mathcal{A} \text{ accepts } t\}$.

Example 8.3. We consider the tree language $T = \{t \in T_{\{a,b\}} \mid \text{there is a path } \pi \text{ through } t \text{ such that } t|\pi \in (a+b)^*(ab)^\omega\}$. The language can be recognized by a Muller tree automaton $\mathcal{A}$ that guesses a path through t and checks, if the label of this path belongs to $(a+b)^*(ab)^\omega$. For this purpose $\mathcal{A}$ memorizes in its state the last read input symbol. If in the next step the current input symbol varies from that in the state memory, then it gets noticed in $\mathcal{A}$'s successor state, otherwise $\mathcal{A}$ switches back to the initial state q_I. Hence a path label in $(a+b)^*(ab)^\omega$ involves an infinite alternation between a state q_a memorizing input symbol a and a state q_b memorizing b. Therefore $\mathcal{F}$ includes the acceptance set $\{q_a, q_b\}$. It remains to be explained how $\mathcal{A}$ can guess a path. Guessing a path means to decide whether the left or the right successor node of the input tree belongs to the path. In the corresponding run this node obtains the label q_a or q_b, depending on the current input symbol. The remaining node gets the label q_d which signals that it is outside the guessed path.

Formally, $\mathcal{A} = (\{q_\mathrm{I}, q_a, q_b, q_d\}, \{a, b\}, \Delta, q_\mathrm{I}, \{\{q_a, q_b\}, \{q_d\}\})$. Transition relation Δ includes the following initial transitions $(q_\mathrm{I}, a, q_a, q_d)$, $(q_\mathrm{I}, a, q_d, q_a)$,

$(q_\mathrm{I}, b, q_b, q_d)$, $(q_\mathrm{I}, b, q_d, q_b)$. Since we do not care about the situation outside the path guessed, i.e, in a run the left and right successors of a node labeled by q_d will get the label q_d as well, independently of the current input symbol, it follows $(q_d, a, q_d, q_d) \in \Delta$ and $(q_d, b, q_d, q_d) \in \Delta$. If for a node with label q_a the corresponding input label is b, then the automaton enters state q_b, formally (q_a, b, q_b, q_d), $(q_a, b, q_d, q_b) \in \Delta$. Reading an a instead means that there have been two consecutive a's, i.e., we are still checking the label prefix $(a + b)^*$. In this case $\mathcal{A}$ reenters q_I, that is, $(q_a, a, q_\mathrm{I}, q_d)$,$(q_a, a, q_d, q_\mathrm{I}) \in \Delta$. Since the case for node label q_b is symmetrical, (q_b, a, q_a, q_d),$(q_b, a, q_d, q_a) \in \Delta$ and $(q_b, b, q_\mathrm{I}, q_d)$,$(q_b, b, q_d, q_\mathrm{I}) \in \Delta$.

On the input tree t of Example 8.1 there exists a successful run ϱ that could start with the transitions depicted in Figure 8.2.

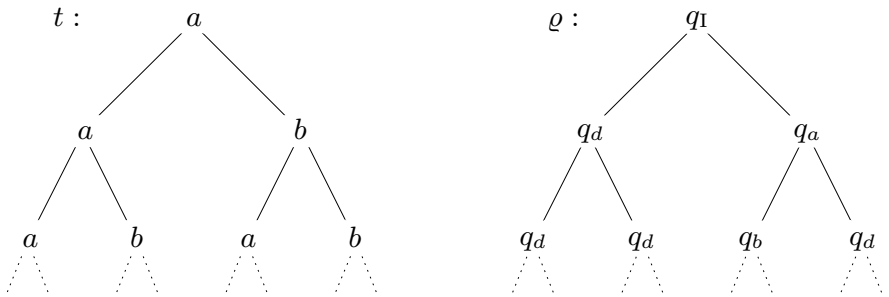

Fig. 8.2. First transitions of ϱ

Exercise 8.2. Define a Muller tree automaton recognizing the language $T = \{t \in T_{\{a,b\}} \mid$ there is a path π through t such that after any occurrence of letter a in π there is some occurrence of letter $b\}$.

In a similar way we can define **parity tree automata**, that is, we adopt the **parity condition**, introduced in [55], to tree automata. It will turn out that this automaton model is particularly useful for the solution of the complementation problem for automata on infinite trees.

Definition 8.4. A parity tree automaton is a quintuple $\mathcal{A} = (Q, \Sigma, \Delta, q_\mathrm{I}, c)$ where Q is a finite state set, Σ is a finite alphabet, $\Delta \subseteq Q \times \Sigma \times Q \times Q$ denotes the transition relation, q_I is an initial state, and $c : Q \to \{0, ..., k\}$, $k \in \mathbb{N}$ is a function which assigns an index value out of a finite index set to each state of the automaton. Sometimes the index values are called colors where c is the corresponding coloring function. Again, a run of $\mathcal{A}$ on an input tree $t \in T_\Sigma$ is a tree $\varrho \in T_Q$, satisfying $\varrho(\varepsilon) = q_\mathrm{I}$ and $\forall w \in \{0, 1\}^* : (\varrho(w), t(w), \varrho(w0), \varrho(w1)) \in \Delta$. We call it successful if for each path $\pi \in \{0, 1\}^\omega$ the **parity acceptance condition** is satisfied, that is, if $min\{c(q) \mid q \in \mathrm{Inf}(\varrho|\pi)\}$ is even. The tree language recognized by $\mathcal{A}$ is the set $T(\mathcal{A}) = \{t \in T_A^\omega \mid \mathcal{A}$ accepts $t\}$.

Example 8.5. We consider the tree language $T = \{t \in T_{\{a,b\}} \mid$ for each path π through t holds $t|\pi \in a^\omega \cup (a + b)^* b^\omega\}$. The language can be recognized by a

parity tree automaton $\mathcal{A}$ that checks simultaneously whether the labels of all paths belong to $a^\omega \cup (a+b)^* b^\omega$ or not. Hence there is no necessity to guess a correct path, i.e, for each state the left and right successor states will be identical.

The automaton starts in the initial state q_I and changes to successor states q_b, q_b if an alphabet symbol b was read and remains in q_I for a symbol a, respectively. We observe that reading a symbol b means we cannot have a label a^ω on the corresponding path. The following initial transitions $(q_\mathrm{I}, b, q_b, q_b), (q_\mathrm{I}, a, q_\mathrm{I}, q_\mathrm{I})$ belong to the transition relation Δ of $\mathcal{A}$. The automaton remains in q_b if the corresponding input is a b, i.e., $(q_b, b, q_b, q_b) \in \Delta$, otherwise it switches both successor states and thus $(q_b, a, q_a, q_a) \in \Delta$. $\mathcal{A}$ behaves symmetrically when its current state is q_a, that is, $(q_a, a, q_a, q_a), (q_a, b, q_b, q_b) \in \Delta$.

While reading a's, $\mathcal{A}$ labels the nodes of his run on t with q_I. An alphabet symbol b signals that from now on the automaton has to verify $(a+b)^* b^\omega$. This is done by using the states q_a and q_b which indicate that the symbol last read was a or b, respectively. On paths which labels belong to $(a+b)^* b^\omega$ the automaton remains, from some point of time, in state q_b and consumes b's exclusively. Thus, if we index the states by $c(q_a) = 1$ and $c(q_b) = 2 = c(q_\mathrm{I})$, we can ensure that only the desired trees are accepted.

Exercise 8.3. Define a Muller and a parity tree automaton recognizing the language $T = \{t \in T_{\{a,b\}} \mid \text{any path through } t \text{ carries only finitely many } b\}$.

Büchi, **Rabin** and **Streett** tree automata are defined analogously, i.e., we provide the tree automata with a Büchi, Rabin or Streett acceptance condition. For a thorough definition of these acceptance conditions see Chapter 1. Hence a run of one of these automata is successful if and only if for each path of the run the corresponding acceptance condition is satisfied. Büchi tree automata differ from the other automaton models in terms of their generative capacity, i.e., they differ in terms of the language class recognized. We state this fact in the following theorem.

Theorem 8.6. *Büchi tree automata are strictly weaker than Muller tree automata in the sense that there exists a Muller tree automaton recognizable language which is not Büchi tree automaton recognizable [149].*

Proof. The language $T = \{t \in T_{\{a,b\}} \mid \text{any path through } t \text{ carries only finitely many } b\}$ can obviously be recognized by a Muller tree automaton with transitions $(q_\mathrm{I}, a, q_\mathrm{I}, q_\mathrm{I}), (q_1, a, q_\mathrm{I}, q_\mathrm{I}), (q_\mathrm{I}, b, q_1, q_1), (q_1, b, q_1, q_1)$ and the designated set $\mathcal{F} = \{\{q_\mathrm{I}\}\}$. (This solves one part of the above exercise.) However, it can not be recognized by any Büchi tree automaton.

Assume for contradiction that T is recognized by a Büchi tree automaton $\mathcal{B} = (Q, \Sigma, \Delta, q_\mathrm{I}, F)$ such that $card(Q) = n$. Consider the input tree $t_n \in T_{\{a,b\}}$ which has a label b exactly at the nodes $1^+0, 1^+01^+0, \ldots, (1^+0)^n$, i.e., at positions that we reach by choosing the left successor after a sequence of right successors, but only for at most n left choices. It is obvious that $t_n \in T$. Thus there is a successful run ϱ of $\mathcal{B}$ on t_n. On path 1^ω a final state is visited infinitely often, hence there must be a natural number m_0 so that $\varrho(1^{m_0}) \in F$. The same observation holds

for path $1^{m_0}01^{\omega}$ with m_1 and $\varrho(1^{m_0}01^{m_1}) \in F$. Proceeding in this way we obtain $n+1$ positions $1^{m_0}, 1^{m_0}01^{m_1}, \ldots, 1^{m_0}01^{m_1}0\ldots1^{m_n}$ on which ϱ runs through a final state. This means that there must be positions, say u and v, where $u < v$ and $\varrho(u) = \varrho(v) = f \in F$. We consider the finite path π_u in t_n from u to v. By construction this path performs at least one left turn and thus it contains a node with label b. Now we construct another input tree t'_n by infinite repetition of π_u. This tree contains an infinite path which carries infinitely many b's, thus $t'_n \notin T$, but we can easily construct a successful run on t'_n by copying the actions of ϱ to π_u infinitely often, hence getting a contradiction. □

One can show that Muller, parity, Rabin and Streett tree automata all accept the same class of languages. The proofs are similar to those for sequential automata from the first chapter. This is not a surprising fact because for tree automata the appropriate acceptance condition is applied to each path of a run separately, i.e., to a sequence of states.

Theorem 8.7. *Muller, parity, Rabin and Streett tree automata all recognize the same tree languages.*

Proof. We sketch the transformations of tree automata according to those for word automata described in Chapter 1.

We start with transforming Muller acceptance to parity acceptance. This transformation reuses the modified **LAR construction** already introduced in Section 1.4.2. Let $\mathcal{A} = (\{1, 2, \ldots, n\}, \Sigma, 1, \Delta, \mathcal{F})$ be a Muller tree automaton. The states of the parity tree automaton $\mathcal{A}'$ are permutations of subsets of $\mathcal{A}$'s states together with a marker $\natural$ that indicates the position of the last change in the record. If $(i, a, i', i'') \in \Delta$, then for all states $u\natural v$ where i is the rightmost symbol we have to add transitions $(u\natural v, a, u'\natural v', u''\natural v'')$ to the transition relation set of $\mathcal{A}'$. The states $u'\natural v'$ and $u''\natural v''$ are the successor states determined by the rules described in Section 1.4.2. If the states out of

$$\{\, u\natural v \mid |u| < i \,\} \cup \{\, u\natural v \mid |u| = i \wedge \{\, a \in \Sigma \mid a \sqsubseteq v \,\} \notin \mathcal{F} \,\}$$

are colored by $2i-1$ and the states out of

$$\{\, u\natural v \mid |u| = i \wedge \{\, a \in \Sigma \mid a \sqsubseteq v \,\} \in \mathcal{F} \,\}$$

are colored by $2i$ then $T(\mathcal{A}) = T(\mathcal{A}')$.

Next we transform parity acceptance to a Streett acceptance condition. Let $\mathcal{A} = (Q, \Sigma, \Delta, q_{\mathrm{I}}, c)$ be a parity tree automaton where $c : Q \to \{0, ..., k\}$, $k \in \mathbb{N}$. An equivalent Streett tree automaton is defined by $\mathcal{A}' = (Q, \Sigma, \Delta, q_{\mathrm{I}}, \Omega)$ where $\Omega := \{(E_0, F_0), \ldots, (E_r, F_r)\}$, $r := \lfloor \frac{k}{2} \rfloor$ and for all $i \in \{0, \ldots, r\}$ the sets E_i and F_i are determined by $E_i := \{q \in Q \mid c(q) < 2i+1\}$ and $F_i := \{q \in Q \mid c(q) = 2i+1\}$.

Next, we transform parity acceptance to a Rabin acceptance condition. Let $\mathcal{A} = (Q, \Sigma, \Delta, q_{\mathrm{I}}, c)$ be a parity tree automaton where $c : Q \to \{0, ..., k\}$, $k \in \mathbb{N}$. An equivalent Rabin tree automaton is defined by $\mathcal{A}' = (Q, \Sigma, \Delta, q_{\mathrm{I}}, \Omega)$ where $\Omega := \{(E_0, F_0), \ldots, (E_r, F_r)\}$, $r := \lfloor \frac{k}{2} \rfloor$ and for all $i \in \{0, \ldots, r\}$ the sets

E_i and F_i are determined by $E_i := \{q \in Q \mid c(q) < 2i\}$ and $F_i := \{q \in Q \mid c(q) = 2i\}$.

Next, we transform Streett acceptance to a Muller acceptance condition. Let $\mathcal{A} = (Q, \Sigma, \Delta, q_\mathrm{I}, \Omega)$ be a Streett tree automaton. We define an equivalent Muller tree automaton by $\mathcal{A}' = (\{1, 2, \ldots, n\}, \Sigma, 1, \Delta, \mathcal{F})$ where

$$\mathcal{F} := \{\, G \in \mathscr{P}(Q) \mid \forall (E, F) \in \Omega \,.\, G \cap E \neq \emptyset \vee G \cap F = \emptyset \,\}.$$

Our final transformation transforms Rabin acceptance to Muller acceptance. Let $\mathcal{A} = (Q, \Sigma, \Delta, q_\mathrm{I}, \Omega)$ be a Rabin tree automaton. We define an equivalent Muller tree automaton by $\mathcal{A}' = (\{1, 2, \ldots, n\}, \Sigma, 1, \Delta, \mathcal{F})$ where

$$\mathcal{F} := \{\, G \in \mathscr{P}(Q) \mid \exists (E, F) \in \Omega \,.\, G \cap E \ = \emptyset \wedge G \cap F \neq \emptyset \,\}. \qquad \square$$

Exercise 8.4. Give an example that shows that the straight-forward conversion of Muller ω-automata to Büchi ω-automata from Chapter 1 does not work for tree automata.

8.4 The Complementation Problem for Automata on Infinite Trees

It is not difficult to prove closure under union, intersection and projection for finite tree automata languages. We leave this as an exercise.

Exercise 8.5. Prove closure under union, intersection and projection for the class of Muller tree automaton recognizable languages.

As already mentioned in the introduction, complementation is the essential problem. We will now show closure under complementation for tree languages acceptable by parity tree automata (and hence acceptable by Muller tree automata).

To simplify the proof we use a game-theoretical approach. We identify a parity tree automaton $\mathcal{A} = (Q, \Sigma, \Delta, q_\mathrm{I}, c)$ and an input tree t with an **infinite two-person game** $\mathcal{G}_{\mathcal{A},t}$ having Player 0 and Player 1 playing the game on t. The rules of the game are the following ones. The Players move alternately. Player 0 starts a game by picking an initial transition from Δ such that the alphabet symbol of this transition equals that at the root of t. Player 1 determines whether to proceed with the left or the right successor. His opponent reacts by again selecting a transition from Δ where the alphabet symbol now must equal the input symbol of the left or right successor node in t and the current transition state has to match the left or right successor state of the previous transition, depending on Player 1's selection. So in general, it is the task of Player 0 to pick transitions and it is the task of Player 1 to determine a direction. Hence, due to Gurevich and Harrington [77], Player 0 and Player 1 are sometimes called automaton and pathfinder. The sequence of actions represents a **play** of the game and induces an infinite sequence of states visited along the path across

t. Player 0 **wins** the play if this infinite state sequence satisfies the acceptance condition of $\mathcal{A}$, otherwise Player 1 wins. Player 0's goal is it to show that the state sequences for all paths of the corresponding run meet the acceptance condition, i.e., that $\mathcal{A}$ accepts t. Player 1 tries to prevent Player 0 from being the winner, his goal is to verify the existence of a path such that the corresponding state sequence violates the acceptance condition of $\mathcal{A}$, i.e., the rejection of t by $\mathcal{A}$.

Example 8.8. For our input tree t and the parity tree automaton $\mathcal{A}$ introduced in Example 8.5, Figure 8.3 shows the first moves in a play of $\mathcal{G}_{\mathcal{A},t}$. Each arrow is labeled with that player whose decision determines the succeeding position.

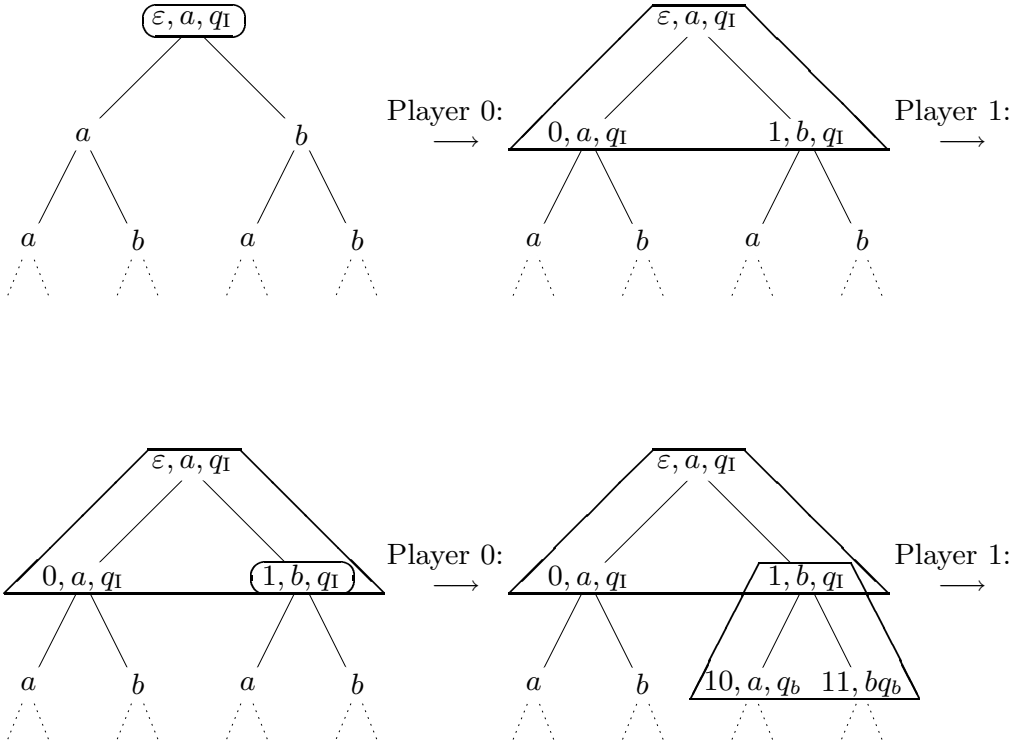

Fig. 8.3. First moves in a play of $\mathcal{G}_{\mathcal{A},t}$

The positions from where on Player 0 or Player 1 have to react are called **game positions**. Thus a play is an infinite sequence of game positions which alternately belong to Player 0 or Player 1. A game can be considered as an **infinite graph** which consists of all game positions as vertices. Edges between different positions indicate that the succeeding position is reachable from the preceding one by a valid action of Player 0 or Player 1, respectively. The game positions of Player 0 are defined by

$$V_0 := \{(w, q) \mid w \in \{0, 1\}^*, q \in Q\}.$$

Player 1's game positions are given by

$$V_1 := \{(w, \tau) \mid w \in \{0,1\}^*, \tau \in \Delta_{t(w)}\},$$

where for each $a \in \Sigma$,

$$\Delta_a := \{\tau \in \Delta \mid \exists q, q'_0, q'_1 \in Q, \tau = (q, a, q'_0, q'_1)\}.$$

In a game position $u = (w, q)$, Player 0 chooses a transition $\tau = (q, t(w), q'_0, q'_1)$ and thus determines the states belonging to the successors of w. Furthermore, by this decision a game position $v = (w, \tau)$ of Player 1 is established. The edge (u, v) then represents a valid move of Player 0. Now Player 1 chooses a direction $i \in \{0, 1\}$ and determines from where to proceed, i.e., Player 1 determines wi and thus establishes $u' = (wi, q'_i)$ which is again a game position of Player 0. The edge (v, u') represents a valid move of Player 1. The usual starting position of a play is $(\varepsilon, q_\mathrm{I})$ and thus belongs to Player 0. Now we index the game positions with the colors of the states belonging to them, i.e., $c((w, q)) = c(q)$ and $c((w, (q, t(w), q'_0, q'_1))) = c(q)$. The games $\mathcal{G}_{\mathcal{A},t}$ then meet exactly the definition of **min-parity games** given in Chapter 4.

Furthermore the notions of a **strategy**, a **memoryless strategy** and a **winning strategy** as defined in Section 2.4 apply to the games $\mathcal{G}_{\mathcal{A},t}$ as well. A winning strategy of a game $\mathcal{G}_{\mathcal{A},t}$ and a successful run $\varrho \in T_Q$ of the corresponding automaton $\mathcal{A} = (Q, \Sigma, \Delta, q_\mathrm{I}, c)$ are closely related.

The run ϱ keeps track of all transitions that have to be chosen in order to accept the input tree t. For any of the nodes (w, q), $w \in \{0,1\}^*$, $q \in Q$, where $(w0, q'_0)$ and $(w1, q'_1)$ are the immediate successors, we can derive the corresponding transition $\tau = (q, t(w), q'_0, q'_1) \in \Delta$. In other words, we know for each node w in each path π through ϱ which transition to apply. Each of these paths is an infinite sequence of states that corresponds to a particular play of the game $\mathcal{G}_{\mathcal{A},t}$. This play is won by Player 0, since the infinite state sequence is a path of the successful run ϱ. The decisions of Player 1 determine the path generated by the current play. Since ϱ determines for each node and each path the correct transition, Player 0 can always choose the right transition, independently of Player 1's decisions, i.e., Player 0 has a winning strategy. Thus if there exists a successful run of $\mathcal{A}$ on t, then Player 0 has a winning strategy.

Conversely, we can use a winning strategy f_0 for Player 0 in $\mathcal{G}_{\mathcal{A},t}$ to construct a successful run ϱ of $\mathcal{A}$ on t. For each game position (w, q) of Player 0, f_0 determines the correct transition $\tau = (q, t(w), q'_0, q'_1)$. Player 0 must be prepared to proceed at game position $(w0, q'_0)$ or at game position $(w1, q'_1)$ since he can not predict Player 1's decision. However, for both positions the winning strategy can determine correct transitions such that the play can be continued to a winning play for Player 0. Hence in ϱ we label w by q, $w0$ by q'_0 and $w1$ by q'_1. Proceeding in this way we obtain the entire run ϱ which is successful since it is determined by a winning strategy of Player 0. Thus, if Player 0 has a winning strategy in $\mathcal{G}_{\mathcal{A},t}$, then there exists a successful run of $\mathcal{A}$ on t.

We summarize these observations in the following lemma.

Lemma 8.9. *A tree automaton $\mathcal{A}$ accepts an input tree t if and only if there is a winning strategy for Player* 0 *from position* $(\varepsilon, q_{\mathrm{I}})$ *in the game* $\mathcal{G}_{\mathcal{A},t}$.

As already mentioned, a game $\mathcal{G}_{\mathcal{A},t}$ which is identified with a parity tree automaton $\mathcal{A}$ and an input tree t meets the definition of parity games. So we can make use of central results about parity games. As is done in Theorem 6.6, it can be shown that these games are determined and that memoryless winning strategies suffice to win a game. Thus from any game position in $\mathcal{G}_{\mathcal{A},t}$, either Player 0 or Player 1 has a memoryless winning strategy.

We are now prepared to focus on our original problem, namely the complementation of finite tree automata languages. Given a parity tree automaton $\mathcal{A}$, we have to specify a tree automaton $\mathcal{B}$ that accepts all input trees rejected by $\mathcal{A}$. Rejection means not accepting an input tree t, or in our game theoretical notation, following Lemma 8.9, there is no winning strategy for Player 0 from position $(\varepsilon, q_{\mathrm{I}})$ in the game $\mathcal{G}_{\mathcal{A},t}$. However, the above-mentioned results about parity games guarantee the existence of a memoryless winning strategy starting at $(\varepsilon, q_{\mathrm{I}})$ for Player 1. We will construct an automaton that checks exactly this.

First of all we observe that a memoryless strategy of Player 1 is a function $f\colon \{0,1\}^* \times \Delta \to \{0,1\}$ determining a direction 0 (left successor) or 1 (right successor). But there is a natural isomorphism between such functions and functions $\{0,1\}^* \to (\Delta \to \{0,1\})$, which, by our definition, are trees. So we can identify memoryless strategies for Player 1 and such trees. We call such trees **strategy trees**, and if the corresponding strategy is winning for Player 1 in the game $\mathcal{G}_{\mathcal{A},t}$, we say it is a **winning tree** for t.

Remark 8.10. Let $\mathcal{A}$ be a parity tree automaton and t be an input tree. There exists a winning tree for Player 1 if and only if $\mathcal{A}$ does not accept t.

Given a parity tree automaton $\mathcal{A}$ and an input t we decide whether a tree s is not a winning tree t using an ω-automaton $\mathcal{M}$ with parity acceptance condition that checks for each path π of t and possible move by Player 0 separately whether the acceptance condition of $\mathcal{A}$ is met. If at least once $\mathcal{A}$'s acceptance condition is met, then s cannot be a winning tree for t and vice versa. Clearly, the automaton $\mathcal{M}$ needs to handle all ω-words of the form $u = (s(\varepsilon), t(\varepsilon), \pi_1)(s(\pi_1), t(\pi_1), \pi_2)\ldots$. Let $L(s,t)$ be the language of all these words.

Example 8.11. Consider a path $\pi = 01100\cdots$ through the tree t. An ω-word $u \in L(s,t)$ determined by π could look like the one depicted in Figure 8.4. Here, every box represents a single alphabet symbol.

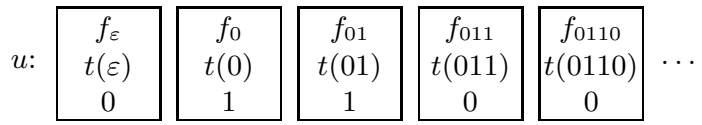

Fig. 8.4. An ω-word determined by π

Let $\mathcal{A}$ be as usual. The automaton $\mathcal{M} = (Q, \Sigma', \Lambda, q_\mathrm{I}, c)$ is designed to handle any trees s and t. So $\mathcal{M}$'s alphabet is defined by $\Sigma' = \{(f, a, i) \mid f\colon \Delta \to \{0,1\}, a \in \Sigma, i \in \{0,1\}\}$. So $\mathcal{A}$ and $\mathcal{M}$ have the same acceptance condition. The automaton $\mathcal{M}$ has to check for each possible move of Player 0 if the outcome is winning for Player 0. This is done nondeterministically: for $(f, a, i) \in \Sigma'$, $f \in map_a$, and $\tau = (q, a, q_0', q_1') \in \Delta_a$ such that $f(\tau) = i$, $\mathcal{M}$ has a transition $(q, (f, a, i), q_i')$. Here, for $a \in \Sigma$, map_a denotes the set of all mappings from Δ_a to $\{0,1\}$.

Lemma 8.12. *The tree s is a winning tree for t if and only if $L(s,t) \cap L(\mathcal{M}) = \emptyset$.*

Proof. "If": Let s be a winning tree. We assume the existence of a path $\pi = \pi_1\pi_2\ldots$ such that the corresponding ω-word

$$u = (s(\varepsilon), t(\varepsilon), \pi_1)(s(\pi_1), t(\pi_1), \pi_2)\ldots$$

determined by π is an element of $L(\mathcal{M})$. So there is a successful run $\varrho = q_\mathrm{I}q_1q_2\ldots$ of $\mathcal{M}$ on u. This implies for each transition

$$(q_j, (s(\pi_1\ldots\pi_j), t(\pi_1\ldots\pi_j), \pi_{j+1}), q_{j+1})$$

that occurs in ϱ the existence of an appropriate transition $\tau_j = (q_j, t(\pi_1\ldots\pi_j), q_0', q_1')$ of $\mathcal{A}$ such that $s(\pi_1\ldots\pi_j) = f_{\pi_1\ldots\pi_j}$ where $f_{\pi_1\ldots\pi_j}(\tau_j) = \pi_{j+1}$. If $\pi_{j+1} = 0$ then $q_{j+1} = q_0'$ otherwise $q_{j+1} = q_1'$ holds. Now we let these transitions τ_j be Player 0's choices in a play of $\mathcal{G}_{\mathcal{A},t}$ where Player 1 reacts by choosing $s(\pi_1\ldots\pi_j)$. The sequence of states visited along this play is $\varrho = q_\mathrm{I}q_1q_2\ldots$ and satisfies $\mathcal{M}$'s acceptance condition. Hence Player 1 loses even though he played according to s. So s cannot be a winning tree for t.

"Only if": Let $L(s,t) \cap L(\mathcal{M}) = \emptyset$. We consider any play of the game $\mathcal{G}_{\mathcal{A},t}$ and assume $(q_j, t(\pi_1\ldots\pi_j), q_0', q_1') \in \Delta$ to be Player 0's choice when $\pi_1\ldots\pi_j$ is the current node. Player 1 plays according to s. The successor state is determined by $s(\pi_1\ldots\pi_j)$ as is described above, i.e., $q_{j+1} \in \{q_0', q_1'\}$. Then we obtain an infinite sequence $\varrho = q_\mathrm{I}q_1q_2\ldots$ of states visited along the play. This sequence is as well the run of $\mathcal{M}$ on the corresponding ω-word $u = (s(\varepsilon), t(\varepsilon), \pi_1)(s(\pi_1), t(\pi_1), \pi_2)\ldots \in L(s,t)$. Since $L(s,t) \cap L(\mathcal{M}) = \emptyset$, ϱ is not accepting. The run ϱ is a particular path of $\mathcal{A}$'s run on t which is determined by Player 0's choices. This implies that $\mathcal{A}$ cannot accept t by this run. However, these observations hold for any run, thus $t \notin T(\mathcal{A})$. □

The word automaton $\mathcal{M}$ accepts all sequences over Σ' which satisfy $\mathcal{A}$'s acceptance condition. However, we are actually interested in a tree automaton $\mathcal{B}$ which recognizes $T(\mathcal{B}) = T^\omega_{\Sigma'} \setminus T(\mathcal{A})$. Thus in order to construct $\mathcal{B}$, we first of all generate a word automaton $\mathcal{S}$ such that $L(\mathcal{S}) = \Sigma' \setminus L(\mathcal{M})$. For this we apply Safra's determinization construction to $\mathcal{M}$ as described in Chapter 3. Actually Safra's algorithm applies to nondeterministic Büchi-automata hence, by the methods specified in Chapter 1, we transform $\mathcal{M}$ to a Büchi-automaton. Now Safra's construction yields a deterministic Rabin automaton that accepts $L(\mathcal{M})$. Since a Streett condition is dual to a Rabin condition, we equip the outcome of

Safra's algorithm with a Streett condition instead of a Rabin condition to obtain the desired word automaton $\mathcal{S} = (Q', \Sigma', \delta, q_{\mathrm{I}}', \Omega)$ such that $L(\mathcal{S}) = \Sigma' \setminus L(\mathcal{M})$. Note that due to the determinization process, the number of $\mathcal{S}$'s states can only be bounded by $2^{O(n \log(n))}$.

Now we are able to construct the desired tree automaton $\mathcal{B} = (Q', \Sigma, \Delta', q_{\mathrm{I}})$, which runs $\mathcal{S}$ in parallel along each path of an input tree. The transition relation of $\mathcal{B}$ is defined by: $(q, a, q_1, q_2) \in \Delta'$ if and only if there exist transitions $\delta(q, (f, a, 0)) = q_1$ and $\delta(q, (f, a, 1)) = q_2$ where $f \in map_a$. Then $T(\mathcal{B})$ accepts $T^{\omega}_{\Sigma'} \setminus T(\mathcal{A})$, as we will prove next.

Theorem 8.13. *The class of languages recognized by finite-state tree automata is closed under complementation.*

Proof. We make use of the constructions given above. It remains to be shown that indeed $T(\mathcal{B}) = T^{\omega}_{\Sigma} \setminus T(\mathcal{A})$.

We assume $t \in T(\mathcal{B})$, i.e., there exists an accepting run ϱ of $\mathcal{B}$ on t. Hence for each path $\pi = \pi_1\pi_2 \cdots \in \{0,1\}^{\omega}$ the corresponding state sequence satisfies Ω and for each node $w \in \{0,1\}^*$ there are transitions $\delta(q, (s(w), t(w), 0) = q_1$ and $\delta(q, (s(w), t(w), 1) = q_2$ of $\mathcal{S}$ where $s(w) \in map_{t(w)}$ and the corresponding transition of $\mathcal{B}$ is $(q, t(w), q_1, q_2)$. This implies that all words $u \in L(s,t)$ are accepted by $\mathcal{S}$ and, since $L(\mathcal{S}) = \Sigma' \setminus L(\mathcal{M})$, $L(s,t) \cap L(\mathcal{M}) = \emptyset$. Due to Lemma 8.12 and Remark 8.10, s is a winning tree for Player 1 and $\mathcal{A}$ does not accept t.

Now let $t \notin T(\mathcal{A})$. This implies the existence of a winning tree s for Player 1 (cf. Lemma 8.10) such that $L(s,t) \cap L(\mathcal{M}) = \emptyset$ (cf. Lemma 8.12) where $\mathcal{M}$ is the nondeterministic word automaton over alphabet Σ' as is constructed above. It follows $L(s,t) \subseteq \mathcal{S}$, i.e., for each path $\pi = \pi_1\pi_2 \cdots \in \{0,1\}^{\omega}$ there exists a run on the ω-word $u = (s(\varepsilon), t(\varepsilon), \pi_1)(s(\pi_1), t(\pi_1), \pi_2) \cdots \in L(s,t)$ that satisfies Ω. Hence by construction of $\mathcal{B}$ there exists an accepting run ϱ of $\mathcal{B}$ on t, that is, $t \in T(\mathcal{B})$. □

Even though the proof of closure under complement is somewhat lengthy due to some technical details, it should be much easier to understand than the original one presented by Rabin [148]. The proof given above highly benefits from a game theoretical view, especially from the observation, that computations of tree automata can be interpreted as parity games. Specifically, it is the determinacy result for this class of games that induces the aforementioned simplification.

8.5 The Emptiness Problem for Automata on Infinite Trees

Beside the closure properties of sets that are recognizable by nondeterministic finite tree automata, algorithmic properties of the automata themselves are of particular interest. In this section, we present an algorithm that decides whether the language accepted by a parity tree automaton is empty or not. Furthermore, we study the complexity of the algorithm.

In order to prove the decidability result we first of all introduce **input-free tree automata**. As the name suggests, this class of tree automata is defined to operate without any input trees. More precisely, an input-free tree automaton is of the form $(Q, \Delta, q_\mathrm{I}, \mathrm{Acc})$ where Q is a finite state set, q_I a designated initial state, $\Delta \subseteq Q \times Q \times Q$ a transition relation, and an acceptance condition. For instance, in case of an input-free parity tree automaton $\mathcal{A}$, a coloring function c would be added. Input-free tree automata can also be defined even without having an acceptance condition. If so, the automata merely consist of Q, a designated initial state and a transition relation $\Delta \subseteq Q \times Q \times Q$.

We call an input-free tree automaton deterministic if and only if for all pairs $(q, q', q''), (q, p', p'') \in \Delta$, $q' = p'$ and $q'' = p''$ holds.

A run of an input-free tree automaton is still a tree $t \in T_Q$, defined in a straightforward manner. If the automaton is deterministic, then t is unique and belongs to a particular class of trees, the so-called regular trees. A tree is called **regular** if and only if it has only a finite number of non-isomorphic subtrees. Formally, this can be defined as follows. Given a tree t and a word $u \in \{0,1\}^*$, let t^u be the tree defined by $t^u(v) = t(uv)$. Then t is called regular if the set $\{\, t^u \mid u \in \{0,1\}^* \,\}$ is finite.

Exercise 8.6. Prove the above claim that the unique run of a deterministic input-free automaton is a regular tree.

Regular trees can be generated by deterministic finite automata via an additional output function with alphabet $\{0,1\}$. Let $\mathcal{A} = (Q, \{0,1\}, \delta, q_\mathrm{I}, f)$ be such an automaton where $f\colon Q \to \Sigma'$ is an additional output function. This automaton generates the tree $t \in T_{\Sigma'}$ defined by $t(w) = f(\delta(q_\mathrm{I}, w))$, i.e., the label at node w is $\mathcal{A}$'s output after it has processed w. Note that the root label $t(\varepsilon)$ is the output of $\mathcal{A}$ in its initial state.

Example 8.14. In Figure 8.5 we present a deterministic finite automaton $\mathcal{A} = (\{q_\mathrm{I}, q_b, q_d\}, \{0,1\}, \delta, f)$, where for each state the output function f has the state's index as output, thus generating the regular tree t.

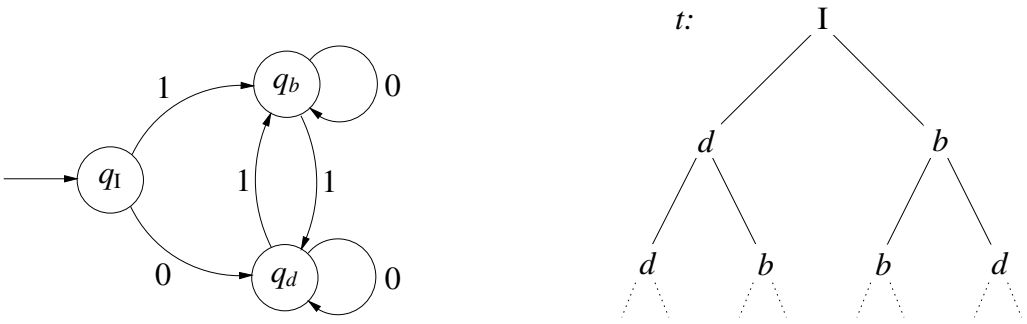

Fig. 8.5. Finite automaton $\mathcal{A}$ generating t

Exercise 8.7. Prove the above claim that a tree is regular if and only if it is generated by a deterministic finite automaton with output function as described above.

Deterministic input-free tree automata without acceptance conditions and deterministic finite-state automata on a binary alphabet are closely related. To see this, we define the state-output pairs $(q, f(q))$ of a deterministic finite automaton $\mathcal{A} = (Q, \{0,1\}, \delta, q_\mathrm{I}, f\colon Q \to \Sigma')$ to be the states of an input-free tree automaton $\mathcal{B} = (Q \times \Sigma', \Delta, (q_\mathrm{I}, f(q_\mathrm{I})))$. Furthermore, we identify the inputs $0, 1$ for $\mathcal{A}$ with the left and right branching of $\mathcal{B}$, i.e., for all $q \in Q$, we let $((q, f(q)), (\delta(q,0), f(\delta(q,0))), (\delta(q,1), f(\delta(q,1)))) \in \Delta$. So $\mathcal{B}$ is deterministic and a run of $\mathcal{B}$ generates in the second component of its states exactly the same tree that $\mathcal{A}$ generates. Hence, in this sense both automaton models have the same expressive power.

Example 8.15. Figure 8.6 presents a run ϱ of the input-free tree automaton $\mathcal{B}$ where $\{(q_\mathrm{I}, \mathrm{I}), (q_b, b), (q_d, d)\}$ is the state set, $\Delta = (((q_\mathrm{I}, \mathrm{I}), (q_d, d), (q_b, b)), ((q_d, d), (q_d, d), (q_b, b)), ((q_b, b), (q_b, b), (q_d, d)))$ and $(q_\mathrm{I}, \mathrm{I})$ is the initial state.

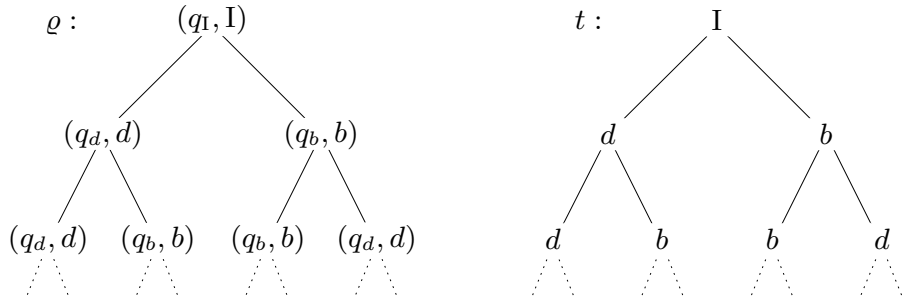

Fig. 8.6. A run ϱ of $\mathcal{B}$ generating t

With respect to the emptiness problem, we now prove the following crucial lemma.

Lemma 8.16. *For each parity tree automaton $\mathcal{A}$ there exists an input-free tree automaton $\mathcal{A}'$ such that $T_\omega(\mathcal{A}) \neq \emptyset$ if and only if $\mathcal{A}'$ admits a successful run.*

Proof. Given a parity tree automaton $\mathcal{A} = (Q, \Sigma, \Delta, q_\mathrm{I}, c)$ we construct an input-free tree automaton $\mathcal{A}' = (Q \times \Sigma, \Delta', \{q_\mathrm{I}\} \times \Sigma, c')$ which has the required property and behaves as follows. $\mathcal{A}'$ guesses an input tree t in the second component of its states nondeterministically. This can be realized by a suitable modification of $\mathcal{A}$'s transition relation. To be more exact, for each transition $(q, a, q', q'') \in \Delta$ we generate transitions $((q,a), (q', x), (q'', y)) \in \Delta'$ if there exist $(q', x, p, p'), (q'', y, r, r') \in \Delta$. Furthermore, for all states of $\mathcal{A}'$ we define $c'(q, a) = c(q)$. So the behavior of $\mathcal{A}'$ on the guessed input t is identical to that of $\mathcal{A}$ running on t. Hence, if $\mathcal{A}'$ has a successful run, then $T_\omega(\mathcal{A}) \neq \emptyset$ and vice versa. □

With every input-free tree automaton $\mathcal{A} = (Q, \Delta, q_I, c)$, we associate a parity game $\mathcal{G}_\mathcal{A}$ which is won by Player 0 if and only if $\mathcal{A}$ has an accepting run. Clearly, we do not have to keep track of input symbols and tree nodes in the corresponding parity game $\mathcal{G}_\mathcal{A}$. The game positions are states from the state set Q of $\mathcal{A}$ and transitions over $Q \times Q \times Q$. More precisely, $V_0 = Q$, $V_1 = \Delta$, and there are two types of transitions. For every $q \in Q$, and $(q, q', q'') \in \Delta$, we have $(q, (q, q', q'')) \in \Delta$; for every $(q, q', q'') \in \Delta$, we have $((q, q', q''), q'), ((q, q', q''), q'') \in \Delta$. The coloring function maps q and (q, q', q'') to $c(q)$.

Clearly, every strategy for Player 0 corresponds to a run and vice versa, and every winning strategy corresponds to a successful run and vice versa.

Remark 8.17. An input-free tree automaton $\mathcal{A}$ admits a successful run if and only if Player 0 wins $\mathcal{G}_\mathcal{A}$.

Example 8.18. Consider an input-free tree automaton with state set $Q = \{q_I, q_a, q_b, q_d\}$, initial state q_I and transition relation $\Delta = \{(q_I, q_a, q_d), (q_I, q_d, q_b), (q_a, q_a, q_I), (q_a, q_d, q_a), (q_d, q_d, q_b), (q_b, q_b, q_d)\}$. The corresponding game graph is depicted in Figure 8.7.

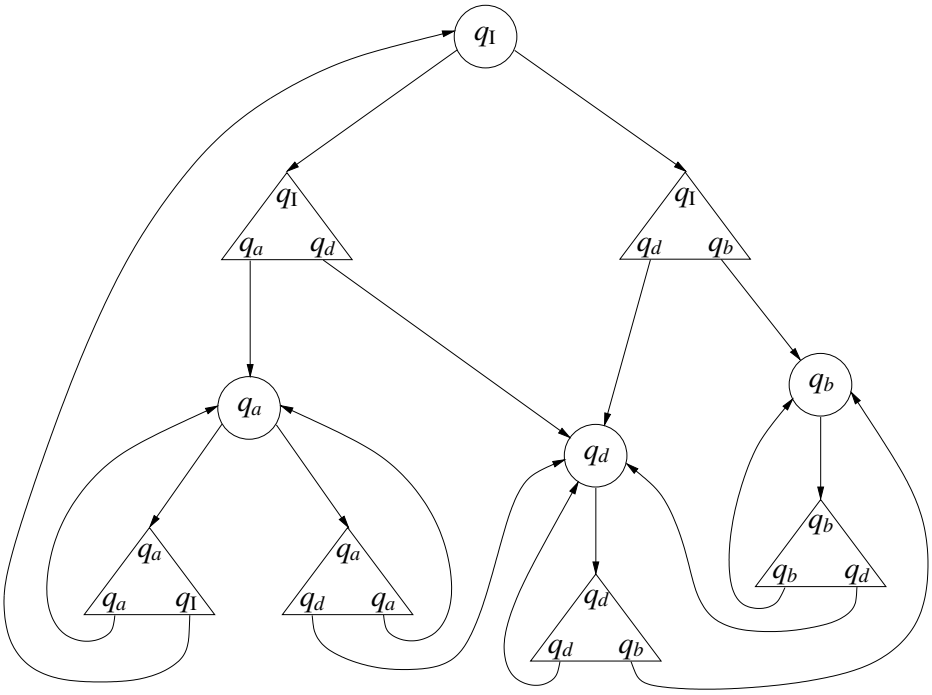

Fig. 8.7. A finite game graph

Since the state set of a tree automaton is finite, the game graph of $\mathcal{G}_\mathcal{A}$ is finite as well and, according to Sections 6.3 and 6.4, the winning strategies for both players are effectively computable. This allows us to solve the emptiness problem.

Theorem 8.19. *For parity tree automata it is decidable whether their recognized language is empty or not.*

Proof. Given a parity tree automaton $\mathcal{A}$, we assume $\mathcal{A}'$ to be an input-free tree automaton that has a successful run iff $T_\omega(\mathcal{A}) \neq \emptyset$. Due to Lemma 8.16, such an automaton exists. Now we identify $\mathcal{A}'$ with the parity game $\mathcal{G}_{\mathcal{A}'}$ and keep in mind that the corresponding game graph is finite because $\mathcal{A}'$ is input-free. From our game-theoretical considerations we know that there is a successful run of $\mathcal{A}'$ if and only if in $\mathcal{G}_{\mathcal{A}'}$ Player 0 wins from some initial position (q_I, a). Since we can effectively compute the winning regions for Player 0 when the game graph is finite, we are able to decide whether there exists a successful run of $\mathcal{A}'$. □

Corollary 8.20. *If the language of a parity tree automaton is not empty, then it contains a regular tree.*

Proof. We let $\mathcal{A}$ and $\mathcal{A}'$ be defined as in the proof of Theorem 8.19. Now we assume to have a successful run of $\mathcal{A}'$ and a memoryless winning strategy for Player 0 in $\mathcal{G}_{\mathcal{A}'}$ from some starting position (q_I, a). This strategy determines a subgraph of the game graph which is in fact a deterministic input-free tree automaton without acceptance condition. To see this, we just extract the transitions out of the subgraph's game positions for Player 1. The tree automaton can be considered as a part of $\mathcal{A}'$ and generates a regular tree in the second component of its states. Clearly, this regular tree is in $T_\omega(\mathcal{A})$ because $\mathcal{A}'$ behaves exactly like $\mathcal{A}$ does. □

Figure 8.8 shows an illustrative example of the situation described in the proof above.

Example 8.21. Consider the finite game graph $\mathcal{G}_{\mathcal{A}'}$ depicted in Figure 8.7. We observe the absence of second components in our illustration; just consider the second entry to be the index of the corresponding state. Furthermore, assume the coloring $c(q_\mathrm{I}, \mathrm{I}) = 1$, $c(q_b, b) = 2$, $c(q_a, a) = 3$ and $c(q_d, d) = 4$. Thus a winning strategy could determine the subgraph emphasized by solid arcs in Figure 8.8. The regular tree generated by the subgraph is the one depicted in Figure 8.6.

To conclude we give time bounds for solving the emptiness problem.

Corollary 8.22. (1) *The emptiness test for parity tree automata can be carried out in time*

$$O\left(d \cdot r^2 m \left(\frac{rn}{\lfloor d/2 \rfloor}\right)^{\lfloor d/2 \rfloor}\right)$$

where $d \geq 2$ is the number of priorities used in the coloring function.
(2) *The emptiness test for parity tree automata is in UP ∩ co-UP.*

Proof. We analyze the proof of Theorem 8.19. Let $\mathcal{A} = (Q, \Sigma, \Delta, q_\mathrm{I}, c)$ be a parity tree automaton. Furthermore, let $|\Delta| = m$, $|Q| = n$, and $|\Sigma| = r$. In a first step we have to construct the input-free tree automaton $\mathcal{A}' = (Q \times \Sigma, \Delta', \{q_\mathrm{I}\} \times \Sigma, c')$. So this automaton has at most rn states with at most $r^2 m$ transitions. Next we identify $\mathcal{A}'$ with the parity game $\mathcal{G}_{\mathcal{A}'}$ and observe that there exist at most $rn + r^2 m$ vertices and at most $3r^2 m$ edges in this game. The last step invokes

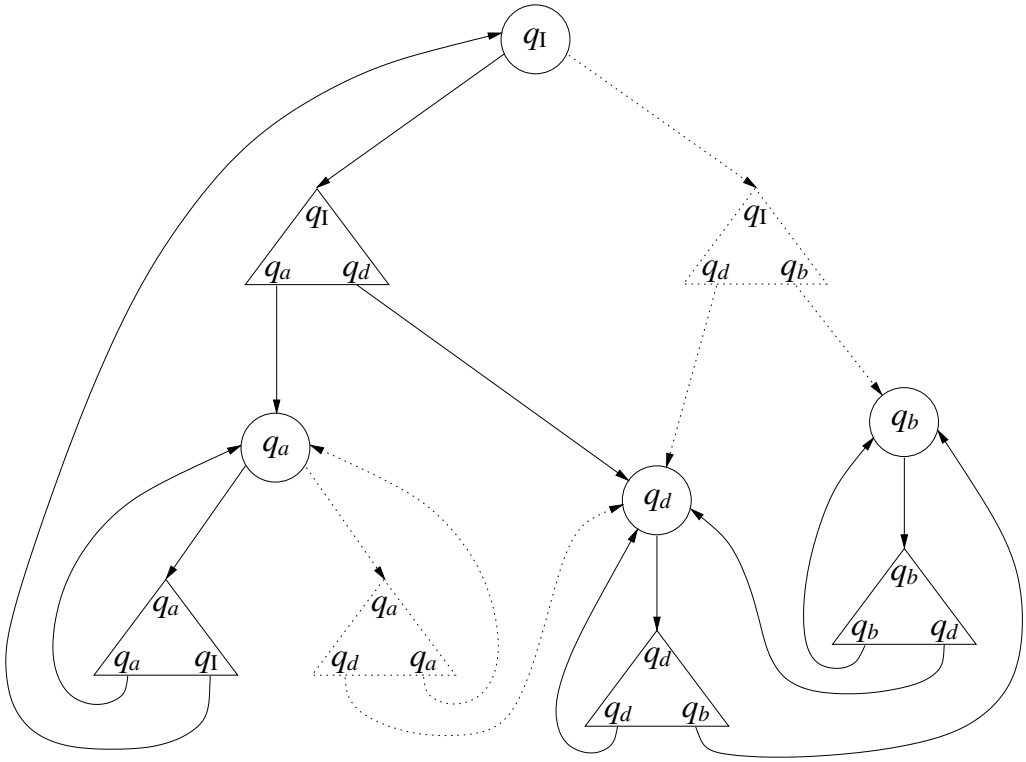

Fig. 8.8. Subgraph determined by Player 0's memoryless winning strategy

an algorithm that computes the winning regions and the winning strategy for Player 0. Here we should apply the best algorithm for the problem known so far (Jurdziński's algorithm [93]) which is thoroughly discussed in Section 7.5. Chapter 6 also presents tight time bounds for this problem, depending on the number of edges, vertices and colors in the game graph. Using this, we get the above bound.

Furthermore, in Chapter 6 it is shown that solving finite parity games lies in the complexity theoretic class UP ∩ co-UP. This proves the second claim. □

Exercise 8.8. Use the above corollary to provide upper bounds for the complexity of the emptiness problem for Rabin tree automata.

8.6 Conclusions

In this chapter we have introduced finite-state automata that are able to consume input trees instead of unidimensional structures. We have applied the acceptance conditions presented in Chapter 1 to our tree automata and have obtained that the resulting models are all equivalent with regard to their acceptance capabilities. Büchi tree automata are an exception; they are weaker, even in their nondeterministic version.

Subsequently we have identified a tree automaton and its input tree with an infinite two-person game. This was significant, since it has allowed us to benefit from various results about infinite games, especially in the proof of closure under

complementation for sets which are recognizable by finite tree automata. This complementation result is essential to prove the decidability of monadic second-order logic and thus demonstrates the importance of tree automaton concepts. More about this will be presented in the Chapter 12.

We have next studied the algorithmic properties of finite tree automata and have shown decidability of the emptiness problem for parity tree automata by again utilizing results about infinite games on finite graphs.