Technische Universität Dresden
Faculty of Computer Science
Institute of Theoretical Computer Science
Chair for Automata Theory

Master's Program in Computer Science

Master's Thesis

Tree Automata with Generalized Transition Relations

submitted by

Sven Dziadek

on September 13th, 2016

Supervisor
Prof. Dr.-Ing. Franz Baader

Advisor
Dr. rer. nat. Daniel Borchmann

Abstract

Finite tree automata consist of a set of states and a set of transitions. Input trees are accepted if it is possible to label the tree with states such that the relationship between the state at every node and the state at its child nodes conforms to a transition. Thus, the transitions only define how a state must be labeled with respect to its children or vice versa.

This work investigates how finite tree automata behave when transitions are made more expressive. Two extensions are considered: a new finite tree automaton model with transitions of arbitrary size and a model where a regular tree language defines the set of transitions. The first extension allows transitions in the shape of trees to have any size. The main difference to conventional tree automata is that larger transitions can overlap. Nevertheless, the new tree automaton model has the same expressive power as conventional tree automata models. The main challenge of the provided proof is eliminating overlappings.

The second extension contains a separate tree automaton called the transition automaton. The language recognized by the transition automaton defines the set of transitions. Consequently, transitions are no longer bounded in size and amount. Nevertheless, this second tree automaton model is not more expressive than conventional models. A constructive proof is provided that uses a conventional tree automaton for simulating the extended tree automaton together with its transition automaton.

Acknowledgment

First, I would like to thank my advisor Dr. Daniel Borchmann who was very helpful. He invested a lot of time to help me when I needed it but he also gave me the time to organize myself. I learned a lot. Thanks for being a good teacher.
I would like to express my gratitude to my supervisor Prof. Franz Baader. He made this thesis possible, and he gave helpful advice and support.
Special thanks to Sarah for motivation and patience, but also for distraction because life is nothing without joy!
I am also thankful to Markus and Ingo for proof-reading and feedback.
Last, I want to thank my friends and my family for support in direct and indirect relation to this work.

Table of Contents

1 Introduction

Finite tree automata are a natural extension to word automata. The key difference between word and tree automata is the alphabet that consists of symbols together with an arity function. The arity function defines for every symbol the number of successors. Words consists only of symbols with arity one and zero. Symbols with multiple successors lead away from linear trees, i.e., words, but towards the "branching" of trees. Similar to automata over words, finite tree automata consist of a finite set of states. An input tree is accepted if it is possible to label the tree with states such that some local conditions are fulfilled. One of the conditions is verified by a transition relation. The tree automaton contains a finite set of transitions and the transitions define how a state must be labeled with respect to its children or vice versa. Additionally, the states for the root and the leaves of the input tree are specified. A tree that can be completely labeled by the transitions is accepted.

As trees are a very common data structure in computer science, tree automata can be used in almost all subfields of computer science. Natural applications are, e.g., compilers processing syntax trees or browser parsing XML documents. Computer linguist process natural languages with special tree automata. Infinite trees are processed in verification. Similar to their counterparts over words, tree automata also provide a basis for theoretic research. Indeed, tree automata have originally been invented to prove the decision problem of monadic second order logic with two successors [TW68; Don70].

To define a specific tree automaton, every transition must be stated explicitly. But sometimes, it is more convenient to define conditions involving more than one node and its children. In this case, the condition has to be split into smaller parts to define transitions that only decide over the states of a node and its children. The goal of this work is to investigate how tree automata change when giving more freedom on how the

transitions have to be defined.

1.1 Contribution

Two extensions of finite tree automata are presented. Both allow more powerful transitions but are similar to conventional tree automata in other aspects.

The first extension is the *general finite tree automaton*. It allows transitions of arbitrary size but the set of transitions must be finite and the transitions must be trees in shape. This automaton model turns out to have the same expressive power as conventional tree automaton models. To show this, we prove that it is possible to decrease the size of the transitions. As larger transitions can overlap, it is necessary to first eliminate overlappings. The elimination of overlappings is the difficult part of the conversion. The proof is constructive and converts every pair of overlapping transitions into a single or a pair of non-overlapping transitions.

The second extension are *regular tree automata*. This automaton model allows a regular tree language as transitions. A tree language is regular if a finite tree automaton recognizes it. Consequently, the regular tree automaton contains a conventional "inner" automaton. This inner automaton accepts trees that are interpreted as transitions in the outer automaton. This has some implications that distinguishes regular tree automata from conventional ones: transitions are no longer bounded in size and the amount of transitions can be infinite. Contrary to intuition, regular tree automata have not more expressive power than general finite tree automata or conventional tree automata. To show this, we simulate a regular tree automaton by a conventional finite tree automaton. As the size of transitions is not bounded, transitions can overlap. The simulating automaton keeps a set of configuration of the "inner" automaton to keep track of all possible transitions at the same time.

1.2 Related Work

Tree automata are an extension to finite automata on words. Pursuing this idea even further, automata on rectangular grids or pictures [BH67; GR92], on partial orderings or acyclic graphs [AG68] and on general graphs [Cou90] have been considered in the literature. Another extension to finite tree automata are tree automata over infinite words [Rab70; Rab72; VW86].

In this section, we present previous work closely related to the topic of this work. The most important contribution is made by Thomas [Tho91]. The definitions of *general* and

regular tree automata in chapter 3 and 4 are based on [Tho91]. After summarizing his work, we shortly present some extensions to his contribution.

Thomas introduces *graph acceptors* in [Tho91]. Graph acceptors are finite automata working on general graphs. His definition of graph acceptors has the same expressive power as monadic second-order logic over graphs. Similar to this work, Thomas allows transitions to be arbitrary subgraphs which are not restricted in size. He presented these results again with another focus in [Tho97b; Tho97a].

A graph acceptor is a triple $\mathcal{A} = (Q, \Delta, C)$ that consists of states Q, transitions Δ and a set of constraints C. The set of transitions is allowed to contain transitions of any size. They are not limited to one node and its neighbors. Constraints C are only needed for the more complex structures like pictures and general graphs.

Thomas gives a theorem with a proof sketch stating that every language accepted by graph acceptors is definable in monadic second-order logic.

Furthermore, a theorem states that tree (or word) languages are recognizable by graph acceptors exactly if they are recognizable by conventional finite automata. The proof is only sketched. One direction is easy because conventional finite automata can be expressed by graph acceptors. [Tho97a] gives slightly more details to the other direction: by the theorem above, a graph acceptor can be transformed into a monadic second-order formula. The conventional finite automaton can be inductively constructed from this monadic second-order formula.

The proof does not work for more complex structures like pictures or general graphs because picture and graph automata are not closed under complement and therefore, in general, it is not possible to express negations of monadic second-order formulae.

As graph acceptors can have arbitrary sized transitions, one can conclude from the theorem above that enlarging transitions for word or tree automata does not give extra expressive power.

The paper further discusses how first-order logic is related to single state graph acceptors and that the monadic theory of context-free graphs is decidable.

The work [Gia+96] proves that transitions in picture automata can be increased in size without increasing the expressive power. It bases on ideas of [Tho91]. The main result of the paper is that their definition of picture automata has the same expressive power as existential monadic second-order logic. One step in the direction from monadic second-order formulae to picture automata shows that transitions can be reduced in size, i.e., (d,d)-tiles can be reduced to (2,2)-tiles.

Latteux and Simplot went another direction: They show in [LS97] that for picture automata, instead of 2x2 tiles, it is equally powerful to use horizontal 1x2 and vertical

2x1 *dominoes*.

1.3 Organization

The presentation is organized as follows: The following chapter 2 defines all preliminaries important for this work. This includes basic notions about trees and also conventional finite tree automata. Chapter 3 discusses the expressive power of tree automata with arbitrary sized transitions. This chapter also proves that the general finite tree automaton model has not more expressive power than conventional tree automaton models. Tree automata with a regular language as transition set are presented in Chapter 4. Their expressive power is discussed in the same chapter. Finally, conclusions are given in Chapter 5.

2 Background

This chapter introduces tree automata as described in the literature. We shall first define alphabets and trees as required for the definitions of tree automata. The second section introduces finite tree automata.

There exist multiple notations for tree automata. The main resources on tree automata used in this work are [TATA; GS15; Baa05; Fin15; Tho90]. We shall loosely follow the notation from [Baa05].

2.1 Basics

In this work we shall only consider trees over ranked alphabets. This includes the special case of words where the arity of every symbol is one except for the last symbol in the word which has arity zero.

A *ranked alphabet* is a tuple (Σ, arity) where Σ is a finite set and arity is a mapping from Σ to $\mathbb{N}$ which denotes the *arity* of a symbol $\sigma \in \Sigma$. Let Σ_p be the set of all symbols in Σ with arity p.

A *finite word* over Σ is the concatenation of finitely many symbols from Σ. $|w|$ is the *length* of the word w. The *empty word* of length zero is denoted by ϵ. The set of all finite words over Σ is denoted by Σ^*. For any word w, the nth symbol of w for $n < |w|$ is denoted by $w(n)$.

Trees are defined as functions over a ranked alphabet.

Definition 2.1 ([Baa05])**.** A *finite* Σ*-tree* is a partial function $t : \mathbb{N}^* \to \Sigma$ whose domain $\text{dom}(t)$ satisfies the *prefix condition*: $\epsilon \in \text{dom}(t)$ and for all $u \in \mathbb{N}^*$ and $i \in \mathbb{N}$ we have

$$ui \in \text{dom}(t) \text{ iff } u \in \text{dom}(t) \text{ and } i < \text{arity}(t(u)).$$

The set of all Σ-trees is denoted by T_Σ. Subsets $L \subseteq T_\Sigma$ are called *tree languages*. ▼

A *leaf* of t is a node $u \in \mathrm{dom}(t)$ such that $\mathrm{arity}(t(u)) = 0$. An *inner node* of t is a node $u \in \mathrm{dom}(t)$ such that $\mathrm{arity}(t(u)) \geq 1$. The *root* of t is the node $u = \epsilon$. A tree t is *finite* if $\mathrm{dom}(t)$ is finite. The *size* of a tree is the number of nodes: $\mathrm{size}(t) = |\mathrm{dom}(t)|$.

A *subtree* of t at position $u \in \mathrm{dom}(t)$ is the tree $t[u]$ with

- $\mathrm{dom}(t[u]) = \{v \mid uv \in \mathrm{dom}(t)\}$
- $t[u](v) = t(uv)$.

Every subtree of t that is not at position ϵ, and is thus not the tree t itself, is called a *proper subtree* of t. Let $\preceq$ be the *prefix relation* on $\mathbb{N}^*$ with $u \preceq v$ iff $\exists u' \in \mathbb{N}^*$ with $uu' = v$. We say $u \prec v$ iff $u \preceq v$ and $u \neq v$. A *path* in t is a maximal and totally ordered subset of $\mathrm{dom}(t)$. The *depth* of a tree is the length of its largest path, i.e., $\mathrm{depth}(t) = \max\{|p| \mid p \in \mathrm{dom}(t)\}$.

2.2 Finite Tree Automata

Tree automata have been introduced to solve the decision problem of the monadic second-order theory of multiple successors [TW68; Don70]. Automata on finite trees can process their inputs either from the leaves to the root or vice versa. Consequently, we call the corresponding automata models *leaf-to-root* and *root-to-leaf automata*.

2.2.1 Leaf-to-Root Automata

The tree automaton processing input trees from the leaves to the root are described in [Don70; TW68]. The automaton is similar to finite automata on words, only the transition relation differs.

Definition 2.2. A *leaf-to-root automaton* $\mathcal{A} = (Q, \Sigma, I, \Delta, F)$ consists of

- a finite set of states Q
- a ranked alphabet Σ
- an initial assignment $I : \Sigma_0 \to 2^Q$
- a transition assignment $\Delta = \{\Delta_a \mid a \in \Sigma\}$, where Δ_a is a function $Q^n \to 2^Q$ for $a \in \Sigma_n$
- a set of final states $F \subseteq Q$.

A *run* on tree $t \in T_\Sigma$ is a mapping $r : \mathrm{dom}(t) \to Q$ such that for all positions $u \in \mathrm{dom}(t)$

$$r(u) \in \Delta_a(r(u0), ..., r(u(n-1))),$$

where $t(u) = a$ and $\mathrm{arity}(a) = n$. A run r is called *successful* iff

- $r(u) \in I(t(u))$ for all leaves u
- $r(\epsilon) \in F$.

An automaton $\mathcal{A} = (Q, \Sigma, I, \Delta, F)$ is said to *accept* $t \in T_\Sigma$ if there is a successful run of $\mathcal{A}$ on t. The language *recognized* by $\mathcal{A}$ is $\mathcal{L}(\mathcal{A}) = \{t \in T_\Sigma \mid \mathcal{A} \text{ accepts } t\}$. ▼

As the automaton is allowed to be nondeterministic, the transition assignment maps tuples of states to sets of states. Alternatively, the transition assignment Δ can also be represented as a relation $\Delta \subseteq Q^n \times \Sigma \times Q$. Analogously, the initial assignment can be seen as a relation $I \subseteq \Sigma_0 \times Q$. The function and the relation representation of Δ and I will be used interchangeably.

2.2.2 Root-to-Leaf Automata

The other direction of finite tree automata, i.e., processing input trees from the root to the leaves, works almost the same, only the transition assignment is "turned around".

Definition 2.3. A *root-to-leaf automaton* $\mathcal{A} = (Q, \Sigma, I, \Delta, F)$ consists of

- a finite set of states Q
- a ranked alphabet Σ
- a set of initial states $I \subseteq Q$
- a transition assignment $\Delta = \{\Delta_a \mid a \in \Sigma\}$, where Δ_a is a function $Q \to 2^{Q^n}$ for $a \in \Sigma_n$
- a final assignment $F : \Sigma_0 \to 2^Q$.

A *run* on the tree $t \in T_\Sigma$ is a mapping $r : \mathrm{dom}(t) \to Q$ such that for all positions $u \in \mathrm{dom}(t)$

$$(r(u0), ..., r(u(n-1))) \in \Delta_a(r(u)),$$

where $t(u) = a$ and $\mathrm{arity}(a) = n$. A run r is called *successful* iff

- $r(\epsilon) \in I$

- $r(u) \in F(t(u))$ for all leaves u.

An automaton $\mathcal{A} = (Q, \Sigma, I, \Delta, F)$ *accepts* $t \in T_\Sigma$ if there is a successful run of $\mathcal{A}$ on t. The language *recognized* by $\mathcal{A}$ is $\mathcal{L}(\mathcal{A}) = \{t \in T_\Sigma \mid \mathcal{A} \text{ accepts } t\}$. ▼

As for leaf-to-root automata, the transition assignment can also be considered as a relation $\Delta \subseteq Q \times \Sigma \times Q^n$.

3

Increasing the Size of Transitions

In this chapter, we shall present a general tree automaton model that allows for arbitrary shapes in its transition relation.

Thomas [Tho91] already extended the transition relation for graph automata as described in Section 1.2. Thomas' paper only gives an overview and does not give details. As almost no proofs are contained in the paper, this chapter adapts his ideas and gives proofs for the relevant parts of this work. Another difference to Thomas' graph automata is that graph automata are not defined to include a *ranked* alphabet as this is not meaningful for graphs. The definitions of tree automata in this work base on ranked alphabets. Even though we changed Thomas' graph automaton considerably, the discussed type of automaton is still strongly inspired by his work.

The first section introduces the notion of general finite tree automata. The next three sections each contain a part of the proof that this new extended automaton is not more expressive than conventional tree automata.

3.1 General Finite Tree Automata

This section introduces a more general kind of finite tree automaton than the automata defined in Section 2.2. The following definition resembles Root-to-Leaf and Leaf-to-Root Automata, but since the transition relation is more general, the direction is not predefined anymore.

In contrary to the original paper [Tho91], the definition of trees follows Section 2.1. Only one extension is needed:

Definition 3.1. Let Σ be a ranked alphabet with arity function and let Q be a non-empty finite set. A *Q-labeled Σ-tree* is a $(Q \times \Sigma)$-tree where $\text{arity}' : Q \times \Sigma \to \mathbb{N}$ is the arity

function derived from the arity function arity of Σ with $\text{arity}'(q, \sigma) = \text{arity}(\sigma)$. ▼

The set of all Q-labeled Σ-trees is denoted by $T_{Q \times \Sigma}$. Note that Q-labeled Σ-trees are still ordinary trees, and so basic definitions from Section 2.1 still apply.

Definition 3.2. A *General Finite Tree Automaton* $\mathcal{A} = (Q, \Sigma, \Delta, A)$ consists of

- a finite set Q of *states*
- a ranked alphabet Σ with arity function
- a finite set of *transitions* Δ containing Q-labeled $(\Sigma \cup \{\Box\})$-trees
- a finite set $A \subseteq Q$ of *accepting states* ▼

$\Box$ is a new symbol of arity zero. The transitions τ are Q-labeled $\Sigma \cup \{\Box\}$-trees. They can also be seen as the product of two separated functions with the same domain: $\tau = r \times t$ with $\tau(p) = (r(p), t(p))$, $r : \text{dom}(\tau) \to Q$ and $t : \text{dom}(\tau) \to \Sigma \cup \{\Box\}$. In this work, transitions τ and pairs (r, t) will be used interchangeably.

We need to define when such an automaton accepts an input tree. Intuitively, the transitions are used to *tile* trees. Only trees that can be tiled in the right way will be accepted. Every node in the tree must be covered by a node in a transition and the special nodes labeled by $\Box$ are used to glue two tiles or transitions together.

An *partial subtree* $\langle t[u] \rangle$ of $t \in T_\Sigma$ at position u results from a subtree $t[u]$ by substituting some of its subtrees by $\Box$.

Definition 3.3. Let $t \in T_\Sigma$ and $u \in \text{dom}(t)$. Let $D \subseteq \text{dom}(t[u])$. The *partial subtree* $\langle t[u] \rangle$ of t at position u is a $(\Sigma \cup \{\Box\})$-tree with domain

$$\text{dom}(\langle t[u] \rangle) = \text{dom}(t[u]) \setminus \{n \in \mathbb{N}^* \mid \exists p \in D. p \prec n\}$$

defined by

$$\langle t[u] \rangle(p) = \begin{cases} \Box, & \text{if } p \in D \\ t[u](p), & \text{otherwise.} \end{cases}$$

A node $p \in \text{dom}(\langle t[u] \rangle)$ is called a *border node* if it is labeled by $\Box$, i.e., $\langle t[u] \rangle(p) = \Box$. Otherwise, it is called a *core node*. ▼

For a tree t to be accepted by an automaton $\mathcal{A}$, every node in the tree must be covered by core nodes of partial subtrees that "correspond" to transitions of $\mathcal{A}$, all of which are compatible with a run on the tree t. Border nodes are used to connect two adjacent partial subtrees.

Definition 3.4. A *run* r of an automaton $\mathcal{A} = (Q, \Sigma, \Delta, A)$ on $t \in T_\Sigma$ is a Q-tree with $\mathrm{dom}(r) = \mathrm{dom}(t)$. The *subrun* $r[u]$ of r at position u is $r[u](p) = r(up)$. ▼

A partial subtree $\langle t[u] \rangle$ of $t \in T_\Sigma$ at position u with a run r on t *matches a transition* $(r', t') \in \Delta$ if $\langle t[u] \rangle = t'$ and $r[u] = r'$. Often, the run r is implicitly clear from the context and will not be stated explicitly.

Definition 3.5. Let r be a run of an automaton $\mathcal{A} = (Q, \Sigma, \Delta, A)$ on a tree $t \in T_\Sigma$. A *tiling* for t with respect to r is a set $\pi = \{t_1, ..., t_n\}$ of partial subtrees of t such that every partial subtree $t_i \in \pi$ matches some transition of Δ and every node of t is the core node of *at least one* partial subtree $t_j \in \pi$. A tiling $\pi = \{t_1, ..., t_n\}$ is called *non-overlapping* or a *NO-tiling* for t if every node of t is the core node of *exactly one* partial subtree $t_j \in \pi$. ▼

Definition 3.6. A run r of $\mathcal{A} = (Q, \Sigma, \Delta, A)$ on tree $t \in T_\Sigma$ is called (non-overlapping or NO-)*successful* if

1. $r(\epsilon) \in A$
2. there exists a (non-overlapping) tiling for t with respect to r.

In this case, the tiling is also called successful. ▼

Note that the definition of partial subtrees technically allows substituting the root of a subtree by $\Box$. The resulting partial subtree would be a graph with only one node: the root labeled by $\Box$. Even though this is allowed, it will not be useful because it does not contain core nodes. Therefore, partial subtrees where the root is labeled by $\Box$ cannot be used for non-redundant tilings of trees.

Example 3.1. Consider the finite field $\mathbb{F}_2$ and variables p and q. We build a finite tree automaton $\mathcal{A} = (Q, \Sigma, \Delta, A)$ with successful runs for all terms over $\mathbb{F}_2$ that result in the value 1 when p is evaluated to 0 and q to 1. The set of states consists of two states corresponding to the two elements in $\mathbb{F}_2$: $Q = \{S_0, S_1\}$. The set of accepting states is a singleton: $A = \{S_1\}$. Σ contains the operators of $\mathbb{F}_2$ together with the variables p and q: $\Sigma = \{p, q, -, +, \cdot\}$. The arity corresponds to the arity of the operation: addition and multiplication are binary, negation is unary and the arity of the variables is zero.

The set of transitions is depicted in Figure 3.1. It contains all combinations of all operators. The pairs $(S_0, +)$ are simply written together as $S_0\,+$. The first two transitions $S_0\ p$ and $S_1\ q$ correspond to initial states in Leaf-to-Root-Automata, they evaluate the variables.

The first tree in Figure 3.2 shows an example of a tree that has a successful run. The second tree in the same figure shows in addition the states of the successful run. The

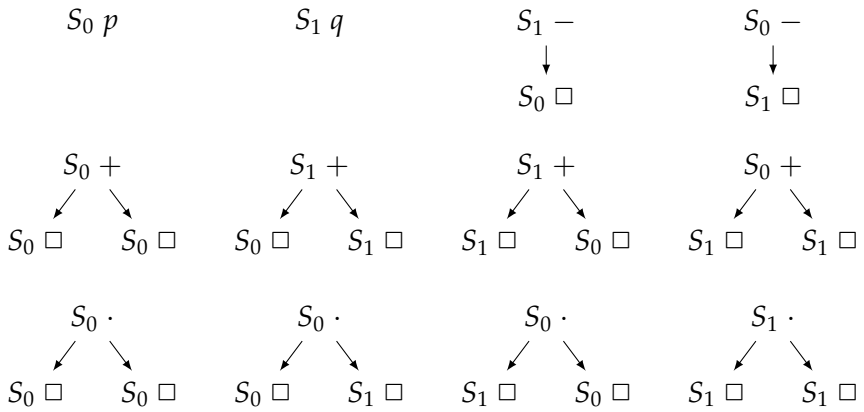


Figure 3.1: Δ of automaton $\mathcal{A}$

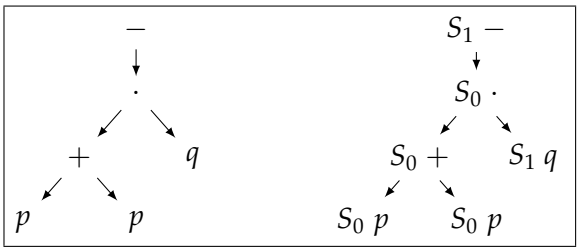


Figure 3.2: Example of an accepted tree and its state labeling

following transitions are used: both single-node transitions, the first negation, the first addition and the second multiplication. All nodes in the tree except for the root are covered by two transitions: once as a core node and once as a border node. ▽

Definition 3.7. A transition $\tau \in \Delta$ is called *elementary* if it contains exactly one core node. If $\mathcal{A}$ only contains elementary transitions, it is called an *elementary* finite tree automaton. ▼

Elementary tree automata only use trees of depth zero and one as transitions. They behave exactly like conventional finite tree automata. Even though their definition is different, it is easy to see that Leaf-to-Root or Root-to-Leaf automata can be translated to elementary finite tree automata and vice versa.

Definition 3.8. $\mathcal{A}$ is said to *(NO-)accept* t if there is a (NO-)successful run of $\mathcal{A}$ on t. Otherwise, $\mathcal{A}$ rejects t. $\mathcal{A} = (Q, \Sigma, \Delta, A)$ *(NO-)recognizes* the language L if for any tree $t \in T_\Sigma$, $t \in L$ iff $\mathcal{A}$ (NO-)accepts t. If there is an automaton $\mathcal{A}$ (NO-)recognizing L, then L is called *(NO-)t-recognizable* or *recognizable by (NO-)tilings*. If $\mathcal{A}$ is elementary, L is called *e-recognizable*. ▼

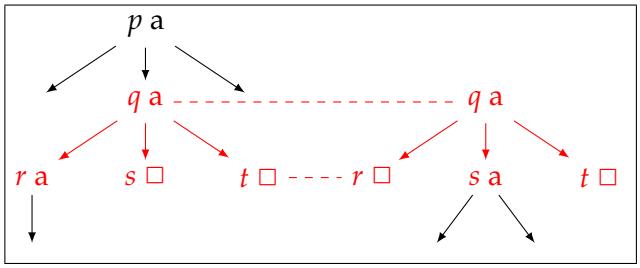


Figure 3.3: Two transitions overlapping

The definition of recognition can also be interpreted the other way around: The language *recognized* by $\mathcal{A} = (Q, \Sigma, \Delta, A)$ is $\mathcal{L}(\mathcal{A}) = \{t \in T_\Sigma \mid \mathcal{A} \text{ accepts } t\}$.

Example 3.2. The tree automaton in the preceding Example 3.1 is elementary because all transitions are elementary. As there is a successful run of $\mathcal{A}$ on the example tree, this tree is accepted by $\mathcal{A}$. It is even NO-accepted because every node is the core node of exactly one transition.

Let L be the language of all terms over $\mathbb{F}_2$ that result in the value 1 when p is evaluated to 0 and q to 1. There are successful runs of $\mathcal{A}$ for every term in L. It follows that $\mathcal{A}$ NO-recognizes L. Consequently, this language is NO-t-recognizable and since $\mathcal{A}$ is elementary, L is even e-recognizable. ▽

It turns out that, in general, every language that is t-recognizable is also NO-t-recognizable and e-recognizable. We shall show this in the following three sections.

3.2 From Overlapping to Non-Overlapping Transitions

Finite tree automata can have overlapping transitions. To transform general tree automata to elementary ones, the first step is to transform the overlapping transitions into new transitions that allow to transform successful runs to NO-successful runs.

Intuitively, overlapping means that a part including the root of one transition is equal to a subtree of the other transition. Furthermore, this part must contain at least one core node of both transitions.

Figure 3.3 shows two transitions overlapping. The two transitions are called *upper* transition (in the figure left) and *lower* transition (in the figure the right transition). As marked by the colors, every transition is divided into two parts: the *matching part* overlaps the other transition (red) and the *individual part* comprises the nodes which are not existing in the other transition (black). The lower transition always contains an

individual part below the matching part. There are three cases to distinguish for the upper transition:

1. In the figure, the upper transition contains an individual part above and one below the matching part.

2. It can happen that the individual part below the matching part is missing. In this case, all individual subtrees belong to the lower transition.

3. The transitions only differ in their subtrees and the individual part above the matching part is missing. Thus, the position $u \in \mathrm{dom}(t_i)$ where the two transitions are overlapping is ϵ.

The following definition formalizes the notion of overlapping and the individual and matching part of overlapping transitions.

Definition 3.9. Two transitions $\tau_1 = (r_1, t_1)$ and $\tau_2 = (r_2, t_2) \in \Delta$ are *overlapping* at position $u \in \mathrm{dom}(t_1)$ if $t_1(u) \neq \square$ and $t_2(\epsilon) \neq \square$ and for all $v \in \mathbb{N}^*$ with $uv \in \mathrm{dom}(t_1)$ and $v \in \mathrm{dom}(t_2)$ either $t_1(uv) \in \{t_2(v), \square\}$ or $t_2(v) = \square$, and $r_1(uv) = r_2(v)$.

τ_1 is called the *upper* and τ_2 the *lower* transition.

t_1 restricted to $\{p \in \mathrm{dom}(t_1) \mid u \not\leq p\}$ is called the *upper individual part* of t_1. $t_1[u]$ restricted to $\mathrm{dom}(t_2)$ is the *matching part* of t_1. The matching part of t_2 is t_2 restricted to $\mathrm{dom}(t_1[u])$. The *individual subtrees* of t_2 are all subtrees $t_2[p]$ for which $t_1[u](p) = \square$ and the *individual subtrees* of t_1 are all subtrees $t_1[up]$ for which $t_2(p) = \square$. ▼

In the following, we want to illustrate by means of some examples how to remove overlappings. The general idea is to *combine* transitions into new transitions that can be included in a non-overlapping tiling. Thus, combining denotes an operation taking two overlapping transitions as input and resulting in one or multiple new transitions that are no longer overlapping but provide the same core nodes for the tiling.

The first example shows a simple way that does not work in general. The next two examples remove overlappings for different cases.

For the next examples, we will often use an alphabet $\Sigma = \{0, 1, 2, ...\}$ where the arity of a symbol corresponds to the meaning of the symbol, i.e., $\mathrm{arity}(0) = 0$, $\mathrm{arity}(1) = 1$ and so on.

Example 3.3. Let $\mathcal{A} = (Q, \Sigma, \Delta, A)$ be a general finite tree automaton with $\Delta = \{\tau_1\}$, where the transition τ_1 is as shown in Figure 3.4 on the left. Transition τ_1 can tile linear subtrees with the only symbol being 1. In a successful run, the states p and q must alternate.

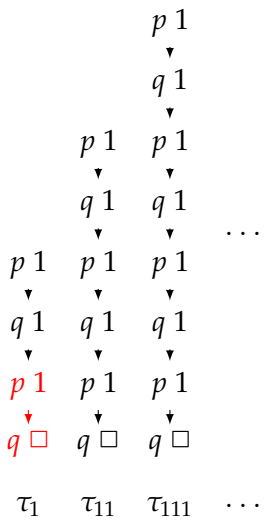


Figure 3.4: Example trees for Example 3.3

Transition τ_1 can overlap with itself. The matching part are the lower two nodes, colored red in the figure. If τ_1 is recursively overlapping multiple times, the transition ensures that there are always an even number of nodes and that the minimum are four nodes.

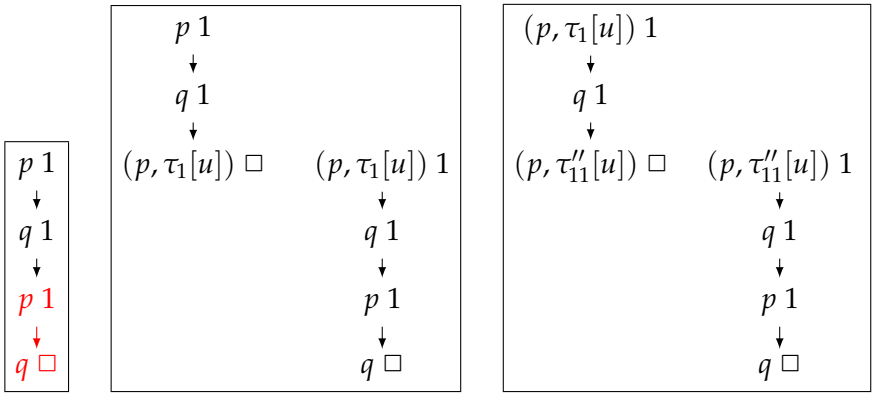
We want to transform $\mathcal{A}$ into an automaton $\mathcal{A}'$ that NO-accepts $\mathcal{L}(\mathcal{A})$. The naive approach is to join overlapping transitions together to one new transition and to explicitly add the joined transitions to the set of new transitions of the new automaton $\mathcal{A}'$. τ_1 is self-overlapping and can be joined with itself. The result is τ_{11} shown in Figure 3.4. As τ_{11} is overlapping with τ_1, these can be joined again resulting in τ_{111} as shown in the same figure. We could continue like this because τ_{111} overlaps with τ_1 and τ_{11} and can be joined with them again.

Formally, $\tau = (r, t)$ overlapping with $\tau' = (r', t')$ at position u can be joined to τ_{new} with domain $\text{dom}(\tau_{\text{new}}) = \text{dom}(\tau) \cup \{uv \mid v \in \text{dom}(\tau')\}$ which is defined as

$$\tau_{\text{new}}(p) = \begin{cases} \tau'(up), & \text{if } p \notin \text{dom}(\tau) \text{ or } t(p) = \square \\ \tau(p), & \text{otherwise.} \end{cases}$$

In the example, the joining of transitions only results in new transitions whose lower two nodes are $p\,1$ and $q\,\square$. Therefore, the new transitions are overlapping with τ_1 and can again be joined with τ_1. This would result in even larger transitions.

But it is not possible to use multiple transitions together in a non-overlapping tiling. To NO-accept the trees that have been accepted by the original automaton with over-



(a) τ_1 (b) τ'_{11} and τ''_{11} (c) τ'_{111} and τ''_{111}

Figure 3.5: Example trees for Example 3.4

lappings, it is necessary to join all possible transitions repeatedly and infinitely. So this results in infinitely many new transitions but the set Δ' of transitions is only allowed to be finite. ▽

As discussed before the previous example, we need a way to combine overlapping transitions into non-overlapping transitions. The approach discussed in the example above does not work for self-overlapping transitions because in an overlapping tiling, τ_1 can be used multiple times but in a non-overlapping tiling, every old or new transition can only be used once and does not fit to the other transitions.

The next two examples show an approach to eliminate overlappings that does work for self-overlapping transitions. This approach is used in the proof below of Theorem 3.2. Instead of joining overlapping transitions together, this approach only removes the matching part of the upper transition. Without the matching part in the upper transition, the two transitions can be used together in a non-overlapping tiling.

Note that it is also possible to remove the matching part of the lower transition instead. But this alternative approach is not very convenient because removing the matching part, including the root, of the lower transition results in a forest of all its individual subtrees. Thus, this approach generates a lot more transitions.

The approach below uses new renamed states to ensure that the shrunk transitions cannot be included in a tiling without the former overlapping partner transition. Thus, the new transitions can only be used together in a tiling.

Example 3.4. The transition τ_1 displayed in Figure 3.5(a) is the same as in Example 3.3. As discussed, this transition overlaps with itself. The matching part has been marked

red in the figure. When combining τ_1 with τ_1, the result are the two transitions τ'_{11} and τ''_{11} displayed in Figure 3.5(b).

As described above, the new transition τ'_{11} does not contain the matching part anymore. Instead, it contains a border node with the renamed state $(p, \tau_1[u])$ where $u = 00$. $\tau_1[u]$ is exactly the removed matching part. The new lower transition τ''_{11} remains the same except for the root which contains the same renamed state $(p, \tau_1[u])$ to ensure that these two transitions can only be used together in a tiling.

The transition τ''_{11} can be further combined with τ_1, the result are the transitions τ'_{111} and τ''_{111} as shown in Figure 3.5(c). As before, the matching part of the upper transition τ''_{11} is removed and substituted by the renamed border node $(p, \tau''_{11}[u])$ $\square$. As discussed, $u = 00$. The lower transition has the same renamed state at the root.

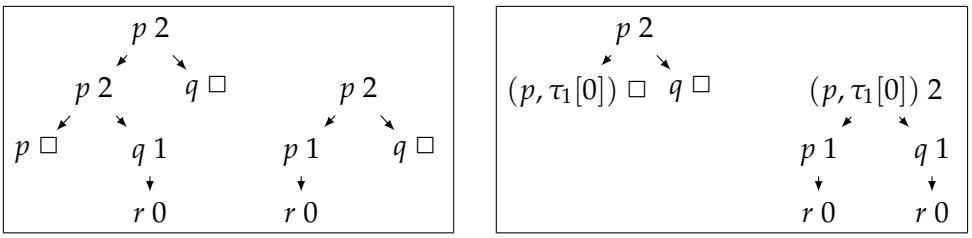
Note that the two renamed states $(p, \tau_1[u])$ and $(p, \tau''_{11}[u])$ are actually equal because the deleted subtrees are the same trees. The transitions τ''_{11} and τ''_{111} are therefore the same. This shows another detail about the renamings: The construction ensures that transitions with renamed root $(q, \chi[u])$ contain the subtree $\chi[u]$ themselves. In a tiling, the renamed border nodes of the upper transitions must be "glued together" with another transition. These states are unique in such a way that they cannot be "glued" to an original transition. But they are allowed to be "glued" to any new transition that contains the same subtree in its root state. In the example, τ'_{11} can be used with τ''_{11}, τ'_{111} and τ''_{111} together in a tiling.

The three new transitions can be used together in a non-overlapping tiling to simulate the original overlapping. The simulation starts with τ'_{11} whose border node is the new state $(p, \tau_1[u]$. From this state, it is possible to use transition τ''_{11} (or τ''_{111}) to end the simulation. Alternatively, the transition τ'_{111} can repeatedly be included in the middle of the tiling to gain trees of arbitrary but even length.

Note that, contrary to the naive approach in the last example, further combinations of these transitions do not yield any new transitions. $\triangledown$

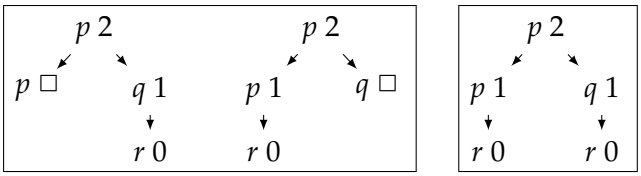
The previous example simplifies the construction in one point: The only individual subtrees in the previous example are contained in the lower transition. In the general case, both the lower and the upper transition can contain individual subtrees. In this case, the individual subtrees of the upper transition are "attached" to the matching part. When removing the matching part, the individual subtrees must be moved to the resulting lower transition. The following example shows how this works.

Example 3.5. Figure 3.6(a) shows two transitions τ_1 and τ_2 overlapping at position 0 of τ_1. Both transitions have individual subtrees. τ_1 provides the right subtree and τ_2 provides the left one.



(a) τ_1 and τ_2 (b) τ'_{12} and τ''_{12}

Figure 3.6: Tree τ_1 and combination τ_{11} for Example 3.5



(a) τ_1 and τ_2 (b) τ'''_{12}

Figure 3.7: Tree τ_1 and combination τ_{11} for Example 3.6

Combining the two transitions τ_1 and τ_2 yields two new transitions τ'_{12} and τ''_{12} depicted in Figure 3.6(b). The resulting upper transition τ'_{12} consists of the upper individual part of τ_1. τ''_{12} consists of the matching part and the individual subtrees of both original transitions. As the new transitions are not allowed to be used individually in a tiling, they have renamed states. ▽

A special case of the previous example is the overlapping of two transitions that have the same root but each transition has different subtrees. In this case, every transition can act as the upper transition. Moving the individual subtrees of one transition to the other transition and removing the matching part of the first transition results in a new upper transition without any nodes. Therefore, this special case produces only one transition created by joining the two input transitions. As there is no "partner", the root of the new transition is not renamed.

Example 3.6. Figure 3.7(a) shows two transitions τ_1 and τ_2 overlapping at position ϵ. Both transitions have individual subtrees but none has an individual part above the matching part.

Combining the two transitions τ_1 and τ_2 yields only one transitions τ'''_{12} depicted in Figure 3.7(b). The resulting transition τ'''_{12} is τ_1 joined with τ_2. No renaming is needed.

Note that the joining discussed in Example 3.3 works in this special case because there is no self-overlapping. But overlapping at the root never comprises self-overlapping. ▽

3.2.1 Construction

Before we prove that we can eliminate overlappings in the general case, we formalize the previous construction. The construction combines overlapping transitions into a set of non-overlapping new transitions. Every pair of overlapping transitions including the newly created ones are combined once. For some pairs of transitions, there might be more than one way to combine them. The combination depends on which is the upper and which is the lower transition and also on what position they will be combined.

Let the binary operator $\otimes_u$ denote the combination of two transitions. The position u is part of the domain of the first operand that also acts as the upper transition. Let $\tau_1, \tau_2 \in T_{Q\times(\Sigma\cup\{\square\})}$. τ_1 and τ_2 can be combined into the set of new transitions $\tau_1 \oplus_u \tau_2 \subseteq T_{Q'\times(\Sigma\cup\{\square\})}$ as follows:

$$\tau_1 \oplus_u \tau_2 = \begin{cases} \{\chi_3\}, & \text{if } \tau_1 \text{ and } \tau_2 \text{ overlap at position } u = \epsilon \\ \{\chi_1, \chi_2\}, & \text{if } \tau_1 \text{ and } \tau_2 \text{ overlap at position } u \neq \epsilon \\ \emptyset, & \text{otherwise} \end{cases}$$

For overlappings in the first case of the definition of $\oplus_u$, i.e., with the same root, both transitions, τ_1 and τ_2 are combined into one new transition χ_3 without renamed states. For overlappings with different roots (in the second case) the transitions χ_1 and χ_2 are constructed by removing the matching part from the upper transition and renaming the new border node of the upper transition and the root of the lower transition to include the deleted subtree.

Let $\tau_1 = (r_1, t_1)$ and $\tau_2 = (r_2, t_2) \in \Delta$ be overlapping at position $u \in \mathrm{dom}(\tau_1)$.

χ_1 is called the *resulting upper transition*. It is created by replacing the matching part of τ_1 with $\square$. As the matching part is the subtree at position u, the new domain is $\mathrm{dom}(\chi_1) = \mathrm{dom}(\tau_1) \setminus \{p \in \mathbb{N} \mid u \prec p\}$. Otherwise, the transition χ_1 equals τ_1. Formally:

$$\chi_1(p) = \begin{cases} (\overbrace{(r_1(p), \tau_1[u])}^{\text{renamed state}}, \square), & \text{if } p = u \\ \tau_1(p), & \text{otherwise} \end{cases}$$

χ_2 is called the *resulting lower transition*. It is a copy of τ_2 where the state of the root is renamed and all individual subtrees of the upper transition τ_1 are added. The new

domain is $\mathrm{dom}(\chi_2) = \mathrm{dom}(\tau_2) \cup \{uv \mid v \in \mathrm{dom}(\tau_1[u])\}$. Formally:

$$\chi_2(p) = \begin{cases} (\overbrace{(r_2(p), \tau_1[u])}^{\text{renamed state}}, t_2(p)), & \text{if } p = \epsilon \\ \tau_1[u](p), & \text{if } \exists v \in \mathrm{dom}(\tau_1[u]) \text{ with } t_2(v) = \square \text{ and } v \preceq p \\ \tau_2(p), & \text{otherwise} \end{cases}$$

When combining two transitions overlapping at the root, the position of the overlapping is $u = \epsilon$. In this case, we just merge the two transitions τ_1 and τ_2 into χ_3. It is build similar to χ_2 with the difference that no states are renamed because there is no "partner" transition produced. The new domain is $\mathrm{dom}(\chi_3) = \mathrm{dom}(\tau_2) \cup \{uv \mid v \in \mathrm{dom}(\tau_1[u])\}$. Formally:

$$\chi_3(p) = \begin{cases} \tau_1[u](p), & \text{if } \exists v \in \mathrm{dom}(\tau_1[u]) \text{ with } t_2(v) = \square \text{ and } v \preceq p \\ \tau_2(p), & \text{otherwise} \end{cases}$$

Additionally, it can happen that three or more transitions in Δ overlap. Finally, self-overlappings are possible. Both possibilities are dealt with by combining the newly created transitions with old transitions in Δ and with the newly created transitions. The final set of new transitions is $\Delta^{\oplus *}$ which is defined as follows:

$$\begin{aligned} \Delta^{\oplus 0} &= \Delta \\ \Delta^{\oplus n+1} &= \Delta^{\oplus n} \cup \bigcup_{\substack{\tau_1, \tau_2 \in \Delta^{\oplus n} \\ u \in \mathrm{dom}(\tau_1)}} \tau_1 \oplus_u \tau_2 \\ \Delta^{\oplus *} &= \bigcup_{n \in \mathbb{N}} \Delta^{\oplus n} \end{aligned}$$

Now, we want to show that $\Delta^{\oplus *}$ is always finite. For this, we need another definition. We can flatten transitions τ with renamed state $(q, \tau'[u])$ at a leaf $v \neq \epsilon$. The resulting transition is $\tau'[u]$ appended to τ at position v.

Definition 3.10. Let $\tau \in \Delta^{\oplus *}$ and let $v \neq \epsilon$ be a leaf of τ. We write $\downarrow_v \tau$ for the transition τ *flattened* at position $v \in \mathrm{dom}(\tau)$. For $\tau(v) = ((q, \tau'[u]), \sigma)$ with $\mathrm{arity}(\sigma) = 0$, the tree $\downarrow_v \tau$ is defined as follows:

$$\mathrm{dom}(\downarrow_v \tau) = \mathrm{dom}(\tau) \cup \{vw \mid w \in \mathrm{dom}(\tau'[u])\}$$

$$\downarrow_v \tau(p) = \begin{cases} \tau(p), & \text{if } v \not\preceq p \\ \tau'[u](q), & \text{if } p = vq \end{cases}$$

Transitions can be *flattened recursively* such that they do not contain any renamed states except at the root. Let $\downarrow\tau$ be the recursively flattened transition τ with the addition that the state of the root is restored to original: $(q, \tau[u])$ is replaced by q. ▼

Proposition 3.1. *$\Delta^{\oplus *}$ is finite.*

Proof. We show that there is some $n \in \mathbb{N}$ such that $\Delta^{\oplus n} = \Delta^{\oplus *}$.

First, we start with some observation. As defined in Section 2.1, the depth of transitions is $\text{depth}(\tau) = \max\{|p| \mid p \in \text{dom}(\tau)\}$. We call a renamed state $(q, \tau[u]) \in Q'$ *nestedly renamed* if the subtree $\tau[u]$ already contains a renamed state.

Now we generalize the depth of a transition to include the depth of the subtrees in the renamed state. We call $\text{depth}(\downarrow t)$ the *nested depth* of transition t.

The combinations $\tau_1 \oplus_u \tau_2$ result, depending on the position u, in different new transitions χ_1, χ_2 and χ_3. As defined above, the domains of these transitions are

$$\begin{aligned} \text{dom}(\chi_1) &= \text{dom}(\tau_1) \setminus \{p \in \mathbb{N} \mid u \prec p\} \\ \text{dom}(\chi_2) &= \text{dom}(\tau_2) \cup \{uv \mid v \in \text{dom}(\tau_1[u])\} \\ \text{dom}(\chi_3) &= \text{dom}(\tau_2) \cup \underbrace{\{uv \mid v \in \text{dom}(\tau_1[u])\}}_{\subseteq \text{dom}(\tau_1)}. \end{aligned}$$

Because of the prefix condition (cf. Definition 2.1), it follows that the depths of the resulting transitions are smaller than or equal to the depth of one of the input transitions.

This result can also be generalized to the nested depth. In comparison to τ_2, χ_2's root is renamed. χ_3 has no new renamed states compared to τ_1 and τ_2. Compared to τ_1, one more border node of χ_1 is renamed to include the deleted subtree of τ_1. It follows that the nested depth of χ_1 equals the nested depth of τ_1. Altogether, the nested depths of the resulting transitions are smaller than or equal to the nested depth of one of the input transitions.

Let $d = \max\{\text{depth}(\tau) \mid \tau \in \Delta\}$. Then $\text{depth}(\downarrow\tau) \leq d$ for all transitions $\tau \in \Delta^{\oplus *}$. As the operation $\downarrow$ only increases the depth, $\text{depth}(\tau) \leq \text{depth}(\downarrow\tau)$ for all transitions $\tau \in \Delta^{\oplus *}$. As the arity is fixed, the size of all new transitions, i.e., the number of nodes, is bounded from above.

Furthermore, $\downarrow\tau$ is a tree in $T_{Q \times (\Sigma \cup \{\square\})}$ because it does not contain any renamed states. Together with the size bound discussed above that means the set $\{\downarrow\tau \mid \tau \in \Delta^{\oplus *}\}$ is finite.

Claim: For every tree $\tau \in T_{Q\times(\Sigma\cup\{\Box\})}$ there are only finitely many trees $\tau' \in \Delta^{\oplus *}$ with $\downarrow\tau' = \tau$. That means the set of pre-images $\downarrow^{-1}(\tau) = \{\tau' \in \Delta^{\oplus *} \mid \downarrow\tau' = \tau\}$ for every transition is finite.

Now we prove the claim. Let $\tau \in T_{Q\times(\Sigma\cup\{\Box\})}$. Let $\tau' \in \downarrow^{-1}(\tau)$. The renamed states at the leaves of τ' contain nested subtrees of transitions. The operator $\downarrow$ flattens these subtrees. All differently nested versions are mapped to one flattened transition. The transition τ is fixed. Thus, the amount of all possible subtrees of τ is bounded. Finally, all possibilities to nest them are bounded.

It is possible that the state at the root of τ' is renamed. Then τ' is the resulting lower transition of the combination $\tau_1 \oplus_u \tau_2$ for some transitions τ_1 and τ_2 and some position $u \neq \epsilon$. The state at the root of τ' is $(q, \tau_1[u])$ with $q = \tau_2(\epsilon)$. From the definition of χ_2 of the construction in Subsection 3.2.1, it follows that the resulting transition τ' contains the subtree $\tau_1[u]$ possibly with more nodes from τ_2 added to it. It follows that the flattened transition τ contains a flattened version of $\tau_1[u]$. The same argument as above holds: τ is fixed. The number of possible parts of τ is bounded. All nested versions of these possible parts are bounded and thus the number of possibilities of subtrees in the renamed state of the root is bounded.
This proves the claim.

Now,

$$\Delta^{\oplus *} = \bigcup\{\downarrow^{-1}(\tau) \mid \tau \in T_{Q\times\Sigma} \text{ with } \mathrm{depth}(\tau) \leq d\}.$$

As the righthand side is finite, so is the lefthand side. It follows that there is an $n \in \mathbb{N}$ such that $\Delta^{\oplus n} = \Delta^{\oplus *}$. □

The following theorem states that we can remove overlappings. Given a general finite tree automaton $\mathcal{A}$, we use the previous construction to define a new general finite tree automaton $\mathcal{A}'$ that NO-recognizes the language $\mathcal{L}(\mathcal{A})$.

Theorem 3.2 (Overlappings)**.** *If a tree language L is t-recognizable, then L is also NO-t-recognizable.*

Proof. Let L be a t-recognizable tree language. Let $\mathcal{A} = (Q, \Sigma, \Delta, A)$ be a finite tree automaton recognizing L.

We construct $\mathcal{A}'$ for which there exists a NO-successful run on every $t \in L$ (Part 1) and for which no successful run on any $t \notin L$ exists (Part 2). Let $\mathcal{A}' = (Q', \Sigma, \Delta', A)$ with $\Delta' = \Delta^{\oplus *}$ be constructed as shown above. Define

$$Q' = Q \cup Q \times \{\tau[u] \mid \tau \in \Delta', u \in \mathrm{dom}(\tau)\}.$$

Part 1: For the following induction, we need a new notion. Let $\pi = \{t_1, ..., t_n\}$ be a successful tiling of automaton $\mathcal{A}$ on a tree t with respect to the run r. Let $v \in \mathrm{dom}(t)$ be a node of the tree t. Let α be the amount of partial subtrees $t_i \in \pi$ that contain v as a core node. Then, the *overlapping degree of a node* v of t is $\mathrm{OD}(v) = \alpha - 1$.

For a successful run, every node is the core node of at least one partial subtree. The overlapping degree denotes the amount of additional partial subtrees covering the node. A degree other than zero signifies overlapping transitions.

The *overlapping degree* $\mathrm{OD}(\pi)$ *of the tiling* π for tree t is the amount of nodes $v \in \mathrm{dom}(t)$ with $\mathrm{OD}(v) > 0$. A successful tiling with overlapping degree of zero is NO-successful.

We need to show that all trees t that have been accepted by $\mathcal{A}$ are NO-accepted by $\mathcal{A}'$. Let $t \in \mathcal{L}(\mathcal{A})$. Let π be a successful tiling for t. We use induction on the overlapping degree of π to show that there is a successful tiling π' of $\mathcal{A}'$ that is non-overlapping.

The tiling π of $\mathcal{A}$ is also a successful tiling of $\mathcal{A}'$ because Δ' and Q' of $\mathcal{A}'$ are supersets of Δ and Q of $\mathcal{A}$, respectively. The induction starts with the tiling π of $\mathcal{A}'$ whose overlapping degree is positive and reduces this degree in every induction step. The tilings with reduced degrees will make use of the new transitions $\tau \in \Delta^{\oplus *}$.

Induction Hypothesis: A successful tiling π of $\mathcal{A}'$ with overlapping degree n can be transformed into a successful non-overlapping tiling for the automaton $\mathcal{A}'$.

Induction Start: For the overlapping degree $n = 0$, the tiling π is already non-overlapping.

Induction Step: Assume the induction hypothesis is true for all overlapping degrees smaller than n. Let $\pi = \{t_1, ..., t_k\}$ be successful tiling with overlapping degree n. We construct a successful tiling π' with $\mathrm{OD}(\pi') < n$.

As $n \geq 1$, there exists at least one node $u \in \mathrm{dom}(t)$ such that $\mathrm{OD}(u) \geq 1$. Pick one of them which is closest to the root, i.e., such that there exists no other node $v \in \mathrm{dom}(t)$ with $\mathrm{OD}(v) \geq 1$ and $|v| < |u|$. Let $\delta = \mathrm{OD}(u)$.

As the overlapping degree of u is positive, it must be a core node in several partial subtrees in π, all of them overlapping at u. Call these partial subtrees $t'_0, t'_1, ..., t'_\delta$ and the matching transitions $\tau_0, \tau_1, ..., \tau_\delta$. At most one of these transitions can have a root higher than the node u because if two transitions contained the parent of node u, this parent would have a positive overlapping degree and would be closer to the root than u. All other transitions have node u as their root.

Distinguish two cases on how to remove the overlapping:

1. If there is no transition starting above u, all transitions $\tau_0, \tau_1, ..., \tau_\delta$ have the same root. Combining two of these transitions results in one new transition with the same root. This new transition can be further combined with the remaining

transitions resulting in one new transition $\tau''' = \tau_0 \oplus_u \tau_1 \oplus_u \cdots \oplus_u \tau_\delta$.
Let t''' be the partial subtree corresponding to τ'''. Define $\pi' = \pi \setminus \{t'_0, t'_1, ..., t'_\delta\} \cup \{t'''\}$.

2. If there is a transition starting above u, call this transition τ_0. First combine all other transitions with the same root as in the first case. Then combine τ_0 with the resulting transition to obtain two new transitions τ' and τ''.
Let t' and t'' be the partial subtree corresponding to τ' and τ'', respectively. Define $\pi' = \pi \setminus \{t'_0, t'_1, ..., t'_\delta\} \cup \{t', t''\}$.

The resulting tiling π' corresponds to a run that is, depending on the case above, equal to the old run or equal to the old run except for the renamed state at the node u in case 2.

As the overlapping at node u has been removed, there is no other transition containing u as a core node. Any other overlappings take place at nodes further away from the root than u. If these overlappings involved any of the transitions $\tau_0, ..., \tau_\delta$, then they now involve τ''' or τ'', because τ' does only contain core nodes strictly between the root and node u. Other transitions whose root node fit border nodes of $\tau_0, ..., \tau_\delta$ now fit the border nodes of τ''' or τ'' because in case 1, no states have been renamed and in case 2, the renamed state is not part of any other transition. The renamed state in case 2 is only part of the overlapping transitions $\tau_0, ..., \tau_\delta$ because every other transition containing node u as a border node would have the parent of node u as a core node and would therefore be overlapping with τ_0 at a node closer to the root than u.

It follows that π' is a valid tiling. As the state of the root is unchanged, the tiling is still successful. The discussed combinations of transitions are also contained in Δ' of $\mathcal{A}'$.

Node u has overlapping degree zero in the new tiling π'. The transformation has not produced more overlappings. Thus, π' has an overlapping degree strictly smaller than n. By induction hypothesis, the tiling π' can be further transformed into a successful non-overlapping tiling.

This completes the induction and it follows that there is a successful tiling π' of $\mathcal{A}'$ that is non-overlapping. For every successful run of the old automaton, there is now a NO-successful run in the new automaton $\mathcal{A}'$.

Part 2: It is left to show that no trees are accepted by $\mathcal{A}'$ that have not been accepted by $\mathcal{A}$. As the set of accepting states A is the same for $\mathcal{A}$ and $\mathcal{A}'$, we only need to show that the construction of Δ' for $\mathcal{A}'$ does not allow tilings of trees that have not been accepted by $\mathcal{A}$.

Let t be a tree accepted by $\mathcal{A}'$. That means there is a successful run r of t and a successful tiling π with respect to r of $\mathcal{A}'$ for t. This tiling may contain partial subtrees corresponding to new transitions in Δ'. We construct a new tiling π' for t that only contains original transitions, i.e., $\pi' \subseteq \Delta$.

The construction of $\Delta^{\oplus *}$ uses the operation $\oplus_u$ to create new transitions. Let τ_1 and τ_2 be transitions in Δ'. Then $\tau_1 \oplus_u \tau_2$ results in new transitions χ_1, χ_2 or χ_3. We distinguish two cases.

1. $u = \epsilon$ and $\tau_1 \oplus_\epsilon \tau_2 = \{\chi_3\}$
2. $u \neq \epsilon$ and $\tau_1 \oplus_u \tau_2 = \{\chi_1, \chi_2\}$

Let t_1, t_2 and x_3 be the partial subtrees corresponding to the transitions τ_1, τ_2 and χ_3, respectively. In the first case, x_3 contains the same core nodes as t_1 and t_2 together because χ_3 is τ_1 and τ_2 "joined". This joining is done without any renamings (cf. construction in Subsection 3.2.1). This means that x_3 can be substituted by t_1 and t_2 in the tiling without changing the run r. Then $\pi' = \pi \setminus \{x_3\} \cup \{t_1, t_2\}$ is a successful tiling for t and all other partial subtrees in π can still be used because this substitution did not change r.

In the second case, substituting the new partial subtrees by the old ones is not easily possible because it reverts the renaming. There can be another transition μ overlapping with χ_1 or χ_2 at the renamed node. When changing the run to the non-renamed state, μ can no longer be used in the tiling.

To remove partial subtrees originating from case 2, we use a different approach by iterating over the tree t. For every node $v \in \mathrm{dom}(t)$ whose state is renamed, restore the name of the state in r and substitute all transitions in π by the transitions they "originated" from. In the following we describe this approach in more detail.

Let π be a tiling for t. Let $v \in \mathrm{dom}(t)$ be a renamed node, i.e., $r(v) = (q, \tau)$. We construct the new run

$$r'(p) = \begin{cases} q, & \text{if } p = v \\ r(p), & \text{otherwise.} \end{cases}$$

Let $t_1, ..., t_k \in \pi$ be the partial subtrees including v as a node. Let $\tau_1, ..., \tau_k$, respectively, be the corresponding transitions. All transitions τ_i contain the renamed state (q, τ) either at the root or at one of the leaves.

Claim: For every transition τ_i, there exists a pair $\tau_i', \tau_i'' \in \Delta'$ such that $\tau_i \in \tau_i' \oplus_p \tau_i''$, where τ_i' and τ_i'' have been overlapping in the node v. τ_i' and τ_i'' contain one renamed state less than τ_i. The other renamed states in τ_i are still contained in τ_i' and τ_i''.

This paragraph gives an intuition[1] on how the claim can be proven. The transition τ_i was constructed out of original transitions in Δ by applying the operator $\oplus$ repeatedly. If the last application of $\oplus$ combined two transitions τ_i' and τ_i'' overlapping at node v, then the claim is easy to show. But it can happen that the operations $\oplus$ have been applied in a different order. Let the operation that renamed the node v to (q, τ) be the jth operation of $\oplus$ and there were other operations $\oplus$ applied afterwards. In this case, "reverting" the changes made in the jth operation will not result in one of the intermediate results. Instead, it will result in a transition τ_{new} gained from applying the operations $\oplus$ in another order. As all transitions in Δ' have been combined by all transitions in Δ', τ_{new} is also contained in Δ' and this should prove the claim.

It follows that

$$\pi' = \pi \setminus \{\tau_i\}_{1 \leq i \leq k} \cup \{\tau_i', \tau_i''\}_{1 \leq i \leq k}$$

is a successful tiling for t with respect to r' which has one renamed state less than r. Inductively, r' can be changed stepwise until it consists only of original states such that the tiling π' with respect to r' only contains original transitions.

This completes Part 2 and it follows that $\mathcal{A}'$ NO-accepts t iff $t \in L$, i.e., L is NO-t-recognizable. □

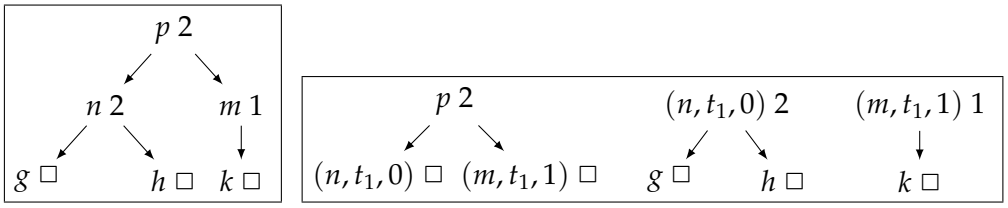
3.3 From Non-Overlapping to Elementary Transitions

Assuming that transitions do not need to overlap to build successful runs, it is easily possible to transform non-elementary transitions into elementary ones. Non-overlapping transitions are connected to other transitions only by matching the state of the border nodes to the state of the root node. Thus, all other states can be renamed. A simple transformation uses this fact by renaming the states of all non-border nodes that are not root nodes to make their states unique. After that, every transition can be split up into elementary parts.

Example 3.7. Transition t_1 in Figure 3.8(a) is not elementary. When transforming t_1 into elementary transitions, the states of all non-border and non-root nodes are renamed. Then the transition can be split into smaller ones. The elementary transitions are shown in Figure 3.8(b). For example, the right transition t_1''' results from cutting the right subtree of t_1 at position 1. ▽

Theorem 3.3. *If tree language L is NO-t-recognizable, then L is also e-recognizable.*

[1] Shortly before handing in this work, the original proof for Part 2 was detected to be imprecise such that this claim will not be exhaustively proven here for time reasons.



(a) t_1 (b) t_1', t_1'' and t_1'''

Figure 3.8: Trees for Example 3.7

The transformation of non-elementary transitions to elementary ones is done by the following construction. The first step is to rename the states of the non-root and non-border nodes. The second step is to separate large transitions into smaller ones.

Let $\tau = (r, t) \in \Delta$ be a transition. Let $\hat{\tau}$ be the transition τ renamed as follows:

$$\hat{\tau}(p) = \begin{cases} \tau(p), & \text{if } p = \epsilon \text{ or } t(p) = \square \\ ((r(p), \tau, p), t(p)), & \text{otherwise} \end{cases}$$

The second step of the transformation is done by an unary operator $\pitchfork_u$ that isolates the elementary part of a transition at position u. Let $\tau = (r, t) \in \Delta$ be a transition. The new transition $\pitchfork_u \tau$ has the domain

$$\mathrm{dom}(\pitchfork_u \tau) = \mathrm{dom}(\tau[u]) \cap (\{\epsilon\} \cup \mathbb{N})$$

and is defined as follows:

$$\pitchfork_u \tau(p) = \begin{cases} \tau[u](p), & \text{if } p = \epsilon \\ (t[u](p), \square), & \text{if } p \in \mathbb{N} \end{cases}$$

The two operators can be applied to the entire set of transitions:

$$\hat{\Delta} = \{\hat{\tau} \mid \tau \in \Delta\} \qquad \text{and} \qquad \pitchfork \Delta = \bigcup_{\tau \in \Delta} \{\pitchfork_u \tau \mid u \in \mathrm{dom}(\tau)\}$$

Finally, the set of elementary transitions is $\pitchfork \hat{\Delta}$.

The following proof used the construction to prove the Theorem. It does not provide any extra arguments but is presented here for completeness.

Proof for Theorem 3.3. Let L be a NO-t-recognizable tree language. Then there exists an automaton $\mathcal{A} = (Q, \Sigma, \Delta, A)$ NO-recognizing L.

We construct an elementary automaton $\mathcal{A}' = (Q', \Sigma, \Delta', A)$ recognizing L. The set of new transitions is constructed as shown above, i.e., $\Delta' = \pitchfork \hat{\Delta}$. Let the combined domain of original transitions be $D = \bigcup_{\tau \in \Delta} \mathrm{dom}(\tau)$. The new set of states is then $Q' = Q \cup (Q \times \Delta \times D)$.

We shall show that $\mathcal{L}(A) = \mathcal{L}(A')$. As $\mathcal{A}$ NO-recognizes L, no overlappings are necessary. That means, in the tiling, the transitions are used such that the border node of one transition matches the root node of another transition. It follows that the states of nodes except the root and the border nodes can be renamed. As a consequence, an automaton using $\hat{\Delta}$ as set of transitions would NO-recognize the same language as $\mathcal{A}$ does.

The second step of the transformation, fulfilled by operator $\pitchfork$, extracts elementary transitions from the renamed transitions in $\hat{\Delta}$. This operation is applied to every node of every transition. These nodes together with their children as border nodes form the new elementary transitions. In this way, it is ensured that every core node in the old transitions is still a core node in the new transitions. Some of the resulting transitions contain only one border node that is also the root node. These transitions cannot be used in the final automaton and can be ignored.

The resulting automaton $\mathcal{A}'$ accepts only trees with the same root as $\mathcal{A}$ because roots have not been renamed and the acceptance condition has not been altered. All transitions in Δ can still be applied because the separated, elementary transitions can be combined to one original, big transition. Also, no separated transition can be used solely because the renamed states are unique and only fit the next elementary transition created in the separation.

It follows that $\mathcal{A}'$ recognizes the same language L as $\mathcal{A}$. As $\mathcal{A}'$ is elementary, L is e-recognizable. □

Corollary 3.4. *For a tree language $L \subseteq T_\Sigma$ the following statements are equivalent:*

1. *L is t-recognizable*
2. *L is e-recognizable*

Proof. The equivalence is shown by two directions. Let L be e-recognizable. From definition, L is also t-recognizable. Now let L be t-recognizable. From Theorem 3.2 follows that L is also NO-t-recognizable. Then from Theorem 3.3, it follows that L is e-recognizable. □

4 Regular Transitions

The tree automata of the previous chapter contained their transitions explicitly in a set Δ. In this chapter, we want to investigate another approach to generalize transitions in tree automata. Its idea is to use an extra automaton to accept transitions, namely to use a root-to-leaf automaton. This automaton model has interesting properties. And it turns out that it does not have more expressive power than root-to-leaf automata.

The first section introduces and defines the mentioned tree automaton model. In the second section, its expressive power is investigated. To prove that the defined regular tree automaton has not more expressive power than finite root-to-leaf automata, a simulation is described and a way to reconstruct the run out of the simulation.

4.1 Regular Tree Automata

The regular tree automaton is in fact similar to the general finite tree automaton. It uses transitions of arbitrary size that are in the shape of partial trees and that are "glued together" by the special symbol $\Box$. But instead of an explicit set of transitions Δ, regular tree automata use an inner transition automaton $\mathcal{A}_\Delta$. All trees accepted by $\mathcal{A}_\Delta$ are interpreted as transitions of $\mathcal{A}$.

Most of the necessary definitions of general tree automata essentially stay valid but are recalled here to adapt them to the new automaton model.

Definition 4.1. A *Regular Tree Automaton* $\mathcal{A} = (Q, \Sigma, \mathcal{A}_\Delta, A)$ consists of

- a finite set Q of *states*
- a ranked alphabet Σ with arity function
- a finite set A of *accepting states*

- a root-to-leaf automaton $\mathcal{A}_\Delta = (Q', Q \times (\Sigma \cup \{\square\}), I, \Delta', F)$, called the *transition automaton* over Q-labeled $(\Sigma \cup \{\square\})$-trees ▼

$\square$ is a new symbol with $\text{arity}(q, \square) = 0$ and $\text{arity}(q, \sigma) = \text{arity}(\sigma)$.

To distinguish transitions in $\mathcal{A}$ and $\mathcal{A}_\Delta$, we call transitions for $\mathcal{A}$ *patterns*. These patterns are trees accepted by $\mathcal{A}_\Delta$ and composed of (elementary) transitions of $\mathcal{A}_\Delta$. Furthermore, we distinguish states of $\mathcal{A}$ and $\mathcal{A}_\Delta$ by calling them $\mathcal{A}$-states and $\mathcal{A}_\Delta$-states respectively. We use the names $q, q', q_i \in Q$ for $\mathcal{A}$-states and $s, s', s_i \in Q'$ for $\mathcal{A}_\Delta$-states.

The definition of *partial subtrees* are independent of the automaton model and remain valid. This definition also distinguishes *border* and *core nodes* that are also used for patterns of regular tree automata.

Definition 4.2. A *run* r of an automaton $\mathcal{A} = (Q, \Sigma, \mathcal{A}_\Delta, A)$ on $t \in T_\Sigma$ is a Q-tree with $\text{dom}(r) = \text{dom}(t)$. ▼

Definition 4.3. Let r be a run of an automaton $\mathcal{A} = (Q, \Sigma, \mathcal{A}_\Delta, A)$ on a tree $t \in T_\Sigma$. A *tiling* for t with respect to r is a set $\pi = \{t_1, ..., t_n\}$ of partial subtrees of t such that every partial subtree $t_i \in \pi$ matches some pattern accepted by $\mathcal{A}_\Delta$ and every node of t is the core node of at least one partial subtree $t_j \in \pi$. ▼

The definition of a tiling requires that the matching patterns are accepted by $\mathcal{A}_\Delta$. This implicitly requires a successful run of $\mathcal{A}_\Delta$ for every used pattern. The definitions of a *successful run*, *acceptance* and *recognition* stay valid:

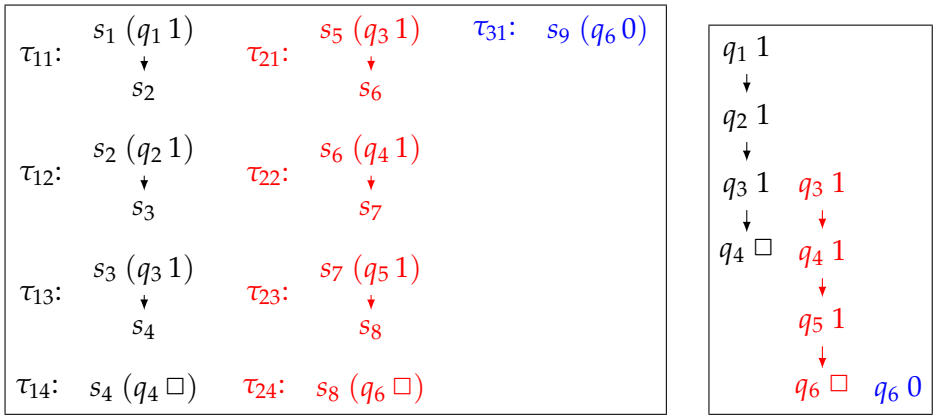
Definition 4.4. A run r of $\mathcal{A} = (Q, \Sigma, \mathcal{A}_\Delta, A)$ on tree $t \in T_\Sigma$ is called *successful* if

1. $r(\epsilon) \in A$
2. there exists a tiling for t with respect to r. ▼

Definition 4.5. $\mathcal{A}$ is said to *accept* t if there is a successful run of $\mathcal{A}$ on t. The automaton $\mathcal{A} = (Q, \Sigma, \mathcal{A}_\Delta, A)$ *recognizes* the language L if for any tree $t \in T_\Sigma$, $t \in L$ iff $\mathcal{A}$ accepts t. The language *recognized* by $\mathcal{A} = (Q, \Sigma, \mathcal{A}_\Delta, A)$ is $\mathcal{L}(\mathcal{A}) = \{t \in T_\Sigma \mid \mathcal{A} \text{ accepts } t\}$. ▼

Note that patterns are no longer limited in size or amount. Instead, the transition automaton can accept infinitely many patterns and these patterns can be arbitrary large. The only requirement is that the set of transitions forms a regular tree language.

As for general finite tree automata, the definition of partial subtrees divides patterns into core and border nodes. The symbol $\square$ is used for border nodes. As patterns are trees accepted by $\mathcal{A}_\Delta$, the transition automaton has to use a different alphabet. More precisely, the alphabet of $\mathcal{A}_\Delta$ consists of pairs of $\mathcal{A}$-states and symbols in $\Sigma \cup \{\square\}$. For

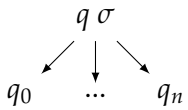


(a) Transitions of $\mathcal{A}_\Delta$ (b) Patterns of $\mathcal{A}$

Figure 4.1: Example trees for Example 4.1

$\mathcal{A}_\Delta$, symbols are pairs where the first element is an $\mathcal{A}$-state and the second is a symbol in Σ or the symbol $\square$.

To simplify our argumentation about the transition automaton, we shall introduce some notation. The transition automaton $\mathcal{A}_\Delta$ is a root-to-leaf automaton. As such, it contains an (inner) transition assignment Δ'. This assignment can be represented as a function or as a relation. As a relation, the assignment is a set of transitions $\tau = (q, \sigma, q_0, ..., q_n) \in Q \times \Sigma \times Q^n$. Every transition can be seen as a tree with a state and a symbol at the root and the children having only states:



The root is at position $\tau(\epsilon) = (q, \sigma)$ and the ith child is at position $\tau(i) = q_i$. This can be generalized to the final assignment. We consider final states $q \in F(\sigma)$ as assignments $(q, \sigma) \in F$. As leaves do not have children, only position ϵ is valid and this position contains the state and the symbol. In the following, we call elements of the final assignment *final transitions*.

Example 4.1. We present a simple regular tree automaton. Let $\mathcal{A} = (Q, \Sigma, \mathcal{A}_\Delta, A)$ be a regular tree automaton, with $Q = \{q_1, q_2, q_3, q_4, q_5, q_6\}$, $\Sigma = \{1, 0\}$ and $A = \{q_1\}$.

The transition automaton is $\mathcal{A}_\Delta = (Q', Q \times (\Sigma \cup \{\square\}), I, \Delta', F)$, where $Q' = \{s_i\}_{1 \leq i \leq 9}$, $I = \{s_1, s_5, s_9\}$ and $F = \{\tau_{14} = (s_4, (q_4, \square)), \tau_{24} = (s_8, (q_6, \square)), \tau_{31} = (s_9, (q_6, 0))\}$. All transitions Δ' together with the final transitions are shown in Figure 4.1(a). The resulting patterns of $\mathcal{A}$ are shown in Figure 4.1(b). Transitions related to the same pattern are

colored in the same color. The only accepted tree is the linear tree with five ones and one zero. ▽

4.2 Expressive Power of Regular Tree Automata

Even though the patterns of the regular tree automaton seem to be much more powerful, the restriction to regular patterns does not gain more expressive power than usual tree automata. This subsection proves that.

The overall structure of the proof is the following: Let $\mathcal{A} = (Q, \Sigma, \mathcal{A}_\Delta, A)$ be a regular tree automaton with $\mathcal{A}_\Delta = (Q', Q \times (\Sigma \cup \{\square\}), I', \Delta', F')$. We construct a root-to-leaf automaton $\mathcal{B} = (Q'', \Sigma, I'', \Delta'', F'')$ that recognizes $\mathcal{L}(\mathcal{A})$. The automaton $\mathcal{B}$ is called a *simulating* tree automaton. A run of $\mathcal{B}$ simulates the run of $\mathcal{A}$ together with all runs of $\mathcal{A}_\Delta$ at once. Recall that there can be several runs of $\mathcal{A}_\Delta$ at the same time because multiple patterns can overlap in the sense of the previous chapter.

Let $D = \{\epsilon\} \cup \{0, ..., k-1\}$ with $k = \max\{\text{arity}(\sigma) \mid \sigma \in \Sigma\}$ be the set of all positions in transitions $\tau \in \Delta'$. A *configuration of* $\mathcal{A}_\Delta$ is a triple $(s, \tau, p) \in Q' \times (\Delta' \cup F') \times D$ consisting of an $\mathcal{A}_\Delta$-state s, a transition τ and a position p.

We call configurations (s, τ, ϵ) with $s \in I'$ *initial configurations*. Configurations (s, τ, ϵ) with $\tau \in F'$ are called *final configurations*. Furthermore, we call configurations (s, τ, ϵ) *epsilon configurations* and configurations (s, τ, p) with $p \neq \epsilon$ are called *non-epsilon configurations*.

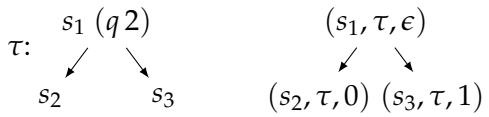
Every final configuration is an epsilon configuration. Configurations do not include a symbol or an $\mathcal{A}$-state. Nevertheless, this information can be associated to epsilon configurations. An epsilon configuration (s, τ, ϵ) is said to *fit* an $\mathcal{A}$-state q or a symbol σ iff $\tau(\epsilon) = (s, (q, \sigma))$.

We further need to be able to speak about how transitions are used in a tiling. A transition $\tau \in \Delta' \cup F'$ is *starting a pattern* if it is used in the run of that pattern as transition for position ϵ. A transition $\tau \in F'$ is *completing a pattern* if it is used in the run of that pattern for a leaf. If transition $\tau_1 \in \Delta'$ is used in a run at position u and $\tau_2 \in \Delta' \cup F'$ is used in this run at position un for some $n \in \mathbb{N}$, then τ_2 is *succeeding* τ_1 and τ_1 is *preceding* τ_2.

We give an example for the above defined notions: The Example 4.1 introduced a set of transitions and a corresponding set of patterns. Both are shown in Figure 4.1 on page 31. τ_{11} is starting a pattern and τ_{14} is completing a pattern. Transition τ_{12} is succeeding transition τ_{11} and τ_{11} is preceding τ_{12}.

In the simulating tree automaton, transitions of the transition automaton are repre-

sented as configurations as follows. A non-final transition τ is a tree of depth one and there exist one corresponding epsilon configuration and at least one corresponding non-epsilon configuration. Here is an example for the arity two:



If transition τ is used in a tiling of a pattern, then the simulation will use the above shown configurations at the corresponding nodes. In a tiling, transition τ will be succeeded by two other transitions with state s_2 or s_3, respectively, at the root. The $\mathcal{B}$-states will therefore always contain at least two configurations for every node v except the root: one epsilon configuration for the transition having v as root and one non-epsilon configuration for the transition having v as child. Additionally, when pattern overlap, there can be more than two configurations in the set of the $\mathcal{B}$-state.

After the introduction of notions, we can now continue the description of the simulating root-to-leaf automaton $\mathcal{B} = (Q'', \Sigma, I'', \Delta'', F'')$. Let t be a tree and let r be a run of $\mathcal{B}$ on t. Let $q \in Q''$ be the state of r at node v of t. q is a pair with the first element being the $\mathcal{A}$-state at this node and the second element being a set of all possible configurations of $\mathcal{A}_\Delta$ at this node. As patterns can overlap, there can be multiple configurations of $\mathcal{A}_\Delta$ for the same node of the input tree. The configurations are therefore arranged in a set. Formally,

$$Q'' = Q \times 2^{Q' \times (\Delta' \cup F') \times D}.$$

The sets I'', Δ'' and the mapping F'' are rather complicated to define and therefore a formal definition will only be given later. We present here only an intuitive description of the sets:

I'' consists of all possibilities of states (q, φ) where $q \in A$ are accepting $\mathcal{A}$-states and φ consists of initial configurations (s, τ, ϵ). All these initial configurations correspond to transitions starting a new pattern. I'' contains multiple states combining multiple possibilities of configurations in every φ. The choice on how many and which initial configurations will be used for the labeling of the root of the input tree is done nondeterministically.

Transitions in Δ'' make sure that non-final epsilon configurations are followed by their corresponding non-epsilon configurations at the child node. They also ensure that every $\mathcal{A}_\Delta$-transition is succeeded by another $\mathcal{A}_\Delta$-transition if the former is not final. Furthermore, they allow new patterns to be started by including initial configurations

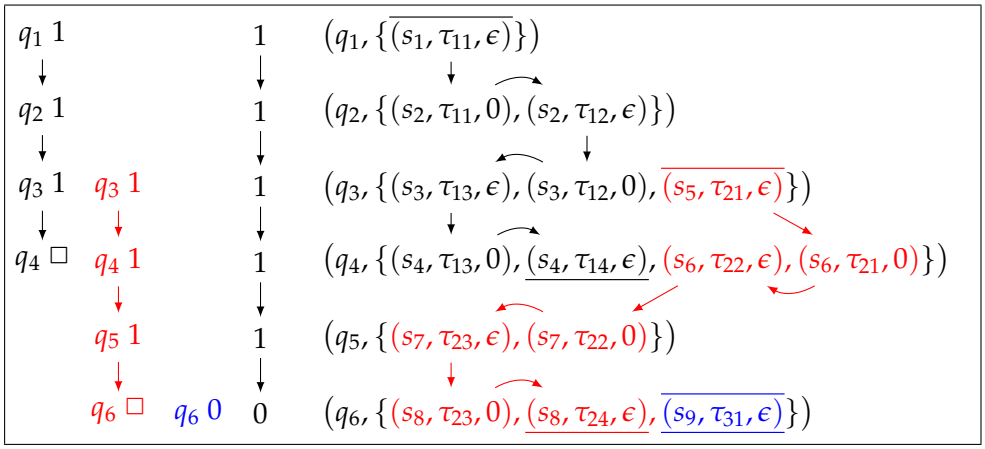


Figure 4.2: Simulation in Example 4.2

in the set of configurations at the $\mathcal{B}$-states of the children. Finally, transitions ensure that all configurations agree on one $\mathcal{A}$-state and on one symbol.

The set $F''(\sigma)$ contains three kinds of configurations: At least one configuration, say (s, τ, ϵ), must be final and the symbol of the contained transition must be σ: $\tau(\epsilon) = (s, (q, \sigma))$. These configuration correspond to the core node at a leaf of a pattern. Additionally, final configurations (s, τ, ϵ) with $\square$ being the symbol of the contained transition $\tau(\epsilon) = (s, (q, \square))$ can be included in every $F''(\sigma)$. They correspond to transitions completing a pattern. Finally, all above mentioned epsilon configurations that are not final must be preceded by other transitions. The non-epsilon configurations of the preceding transitions can also be contained in $F''(\sigma)$.

The following example illustrates the idea used in the main proof. It shows how the configurations correspond to patterns and how the runs of $\mathcal{A}_\Delta$ connects the configurations.

Example 4.2. We construct a new finite root-to-leaf automaton $\mathcal{B} = (Q'', \Sigma, I'', \Delta'', F'')$ simulating $\mathcal{A}$ from Example 4.1. Q is defined as discussed and I'', Δ'' and F'' are described above.

Figure 4.2 shows the accepting run for the only accepted tree of automata $\mathcal{A}$ and $\mathcal{B}$. The lefthand side shows the three patterns used in the tiling and in the middle is the accepted linear tree with six zeros. The righthand side shows a successful run of $\mathcal{B}$. The first values in the tuple of the $\mathcal{B}$-states correspond to the state of $\mathcal{A}$, the set behind it collects some configurations of $\mathcal{A}_\Delta$.

The patterns and the corresponding configurations are displayed in the same color. Additionally, initial configurations are overlined and final configurations are under-

lined. There is also an example of an initial and a final $\mathcal{B}$-state: $(q_1, \{(s_1, \tau_{11}, \epsilon)\}) \in I''$ and $\big(q_6, \{(s_8, \tau_{23}, 0), (s_8, \tau_{24}, \epsilon), (s_9, \tau_{31}, \epsilon)\}\big) \in F''(0)$. The final state contains three configurations, one configuration of every kind in the description of $F''(\sigma)$ above. The last configuration $(s_9, \tau_{31}, \epsilon)$ is needed to make the state a final state. The configuration $(s_8, \tau_{24}, \epsilon)$ completes a pattern and the third configuration is from the transition preceding τ_{24}.

The example also contains five transitions connecting the six $\mathcal{B}$-states. Here are some explanations: Every transition is split into epsilon and non-epsilon configurations. In the figure, they are connected by vertical arrows. Every non-final $\mathcal{A}_\Delta$-transition is succeeded by another $\mathcal{A}_\Delta$-transition. The non-epsilon configuration of the former is together with the epsilon configuration of the latter part of the same $\mathcal{B}$-state. In the figure, they are connected by bent horizontal arrows. As described above for the transition relation Δ'', new patterns are allowed to start at every node. In the example, the red pattern is started at the third node and the blue pattern is started at the last node. ▽

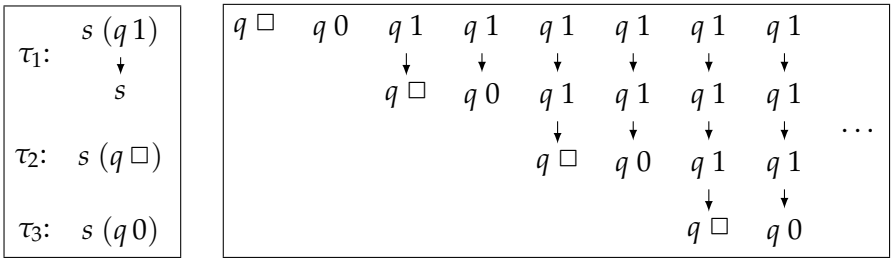
For the simulation, there is one pitfall. The transition automaton $\mathcal{A}_\Delta$ can accept an unbounded number of patterns and it is possible to include arbitrarily many of them in a run. The simulating tree automaton $\mathcal{B}$ uses a finite set of states. This means that many possible runs of the regular tree automaton correspond to one run of $\mathcal{B}$ only.

For a regular tree automaton with unboundedly many patterns, it is not possible to store all overlappings in the simulating root-to-leaf automaton $\mathcal{B}$. But instead of storing patterns, $\mathcal{B}$-states contain a set of configurations corresponding to transitions. As the transition automaton is a finite tree automaton, the set of possible configurations is finite and the resulting set of states Q'' is finite.

If two patterns are overlapping, both can be in the same configuration, i.e., both patterns contain the same transition at the same position. Then it is enough to store only one of these configurations. The intuitive meaning of a configuration in a $\mathcal{B}$-state is that *at least one* run of $\mathcal{A}_\Delta$ is in this configuration at that node. Because of storing only one configuration instead of two, we have to ensure that once more than one successor configuration exists for a given configuration, all these successor configurations can be considered in a run of $\mathcal{B}$ at once. This is to ensure that all possible patterns of $\mathcal{A}$ represented by a single configuration in a $\mathcal{B}$-state are included in a successful run of $\mathcal{B}$. This means that transitions in Δ'' allow for a given non-epsilon configuration to be followed by several epsilon configurations of succeeding transitions.

The next example shows how unbounded overlapping in $\mathcal{A}$ is handled in $\mathcal{B}$.

Example 4.3. Let $\mathcal{A} = (Q, \Sigma, \mathcal{A}_\Delta, A)$ be a regular tree automaton with $Q = \{q\}$, $\Sigma = \{1, 0\}$ and $A = \{q\}$. Let $\mathcal{A}_\Delta = (Q', Q \times (\Sigma \cup \{\square\}), I', \Delta', F')$, with $Q' = \{s\}$, $I' = \{s\}$



(a) Transitions (b) Patterns of $\mathcal{A}$

Figure 4.3: Example trees for Example 4.3

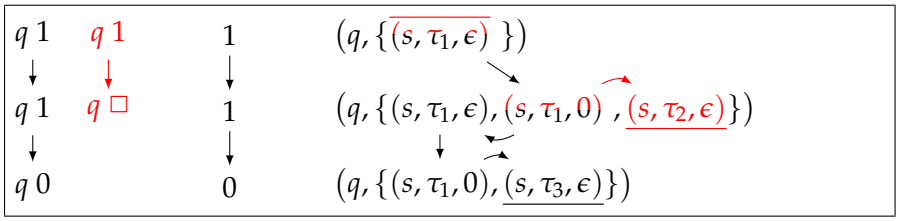


Figure 4.4: Simulation in Example 4.3

and $F' = \{\tau_2 = (s, (q, \square)), \tau_3 = (s, (q, 0))\}$. The only other transition $\tau_1 \in \Delta'$ together with the final transitions are illustrated in Figure 4.3(a).

The transition τ_1 can be succeeded by the two final transitions and by itself. The resulting patterns of $\mathcal{A}$ are shown in Figure 4.3(b). As discussed, the given definition of $\mathcal{A}$ and $\mathcal{A}_\Delta$ allows infinitely many patterns. Except for the final configurations, all configurations for these patterns contain transition τ_1.

Figure 4.4 shows an example of a successful run of $\mathcal{A}$ on the linear tree with two ones and one zero. The accepted tree is shown in the middle. Left of it, there are two patterns of $\mathcal{A}$ used to tile that tree. The right pattern is redundant but is accepted by $\mathcal{A}_\Delta$ and can therefore be included in the tiling. The patterns and their corresponding configurations are colored in the same color. If we consider a successful run for the automaton $\mathcal{B}$, two configurations of $\mathcal{A}_\Delta$ are used by both patterns. These configurations are colored in both colors. As the triples are stored in a set, every triple will be contained at most once.

The configuration $(s, \tau_1, 0)$ of the second node is succeeded by two configurations: (s, τ_1, ϵ) and (s, τ_2, ϵ) (both at the second node). This is the point where the two patterns "diverge" and where both paths are followed at the same time.

The simulation does not distinguish between this tiling and another tiling, both visualized in Figure 4.5. For the simulation, there is no difference, if the second pattern starts at the first or the second node. The corresponding run for both cases is successful

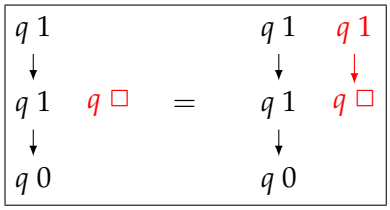
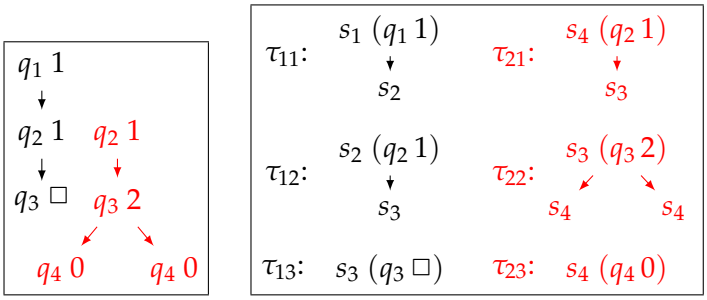


Figure 4.5: Equality of two tilings in Example 4.3



(a) Patterns of $\mathcal{A}$ (b) Transitions of $\mathcal{A}_\Delta$

Figure 4.6: Example trees for Example 4.4

for the same input tree. ▽

Whenever there are unboundedly many patterns, most of these patterns are redundant. They can be left out and can be ignored in the set of configurations in the simulation. For the run of $\mathcal{B}$ to be successful, it is necessary that at least one pattern provides core nodes for the nodes of the input tree. Multiple patterns are not necessary.

Overlapping patterns can have another consequence: it is possible that two patterns overlap in such a way that it is not possible to unambiguously match patterns to configurations. Instead, when reconstructing a run of $\mathcal{A}$, including the patterns accepted by $\mathcal{A}_\Delta$, out of a run of $\mathcal{B}$, there might exist multiple possibilities. The following example illustrates this.

Example 4.4. This example shows a run of $\mathcal{B}$ that can be translated to a run of $\mathcal{A}$ in multiple ways. Let $\mathcal{A} = (Q, \Sigma, \mathcal{A}_\Delta, A)$ with $A = \{q_1\}$, $Q = \{q_1, q_2, q_3, q_4\}$ and $\Sigma = \{0, 1, 2\}$. The transition automaton $\mathcal{A}_\Delta = (Q', Q \times (\Sigma \cup \{\square\}), I, \Delta', F)$ consists of $Q = \{s_i\}_{1 \le i \le 4}$, $I = \{s_1, s_4\}$ and $F = \{\tau_{13}, \tau_{23}\}$. All transitions Δ' together with the final assignments are depicted in Figure 4.6(b). This results in the pattern shown in Figure 4.6(a).

A simulation of $\mathcal{A}$ by a new automaton $\mathcal{B}$ is shown in Figure 4.7. As in the examples before, it shows the used patterns of $\mathcal{A}$ on the left, the accepted tree in the middle and it

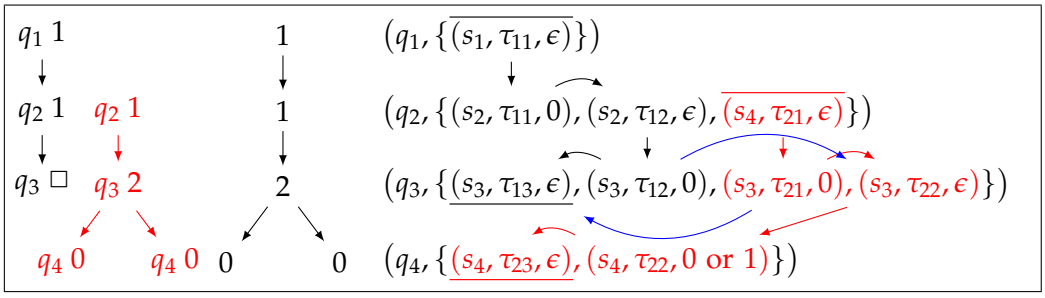
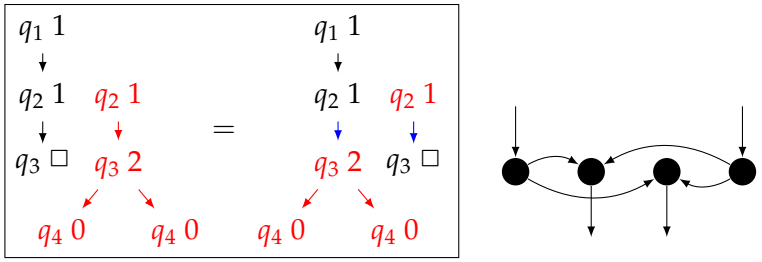


Figure 4.7: Simulation in Example 4.4



(a) Two possible tilings

(b) 2-to-2 connection

Figure 4.8: Tilings and many-to-many connection for Example 4.4

shows the states of the corresponding successful run of $\mathcal{B}$ on the right. There are two leaf nodes but the figure shows only one $\mathcal{B}$-state for both of them. Depending on which node, the configuration (s_4, τ_{22}, p) contains position $p = 0$ or $p = 1$.

The simulation works as seen before. But in contrast to before, there are now two blue arrows connecting configurations at the third level: $(s_3, \tau_{12}, 0) \to (s_3, \tau_{22}, \epsilon)$ and $(s_3, \tau_{21}, 0) \to (s_3, \tau_{12}, \epsilon)$. As discussed, the bent vertical arrows connect succeeding transitions. As both transitions τ_{12} and τ_{21} enforce the $\mathcal{A}_\Delta$-state s_3 at the child, they both can be succeeded by τ_{13} and τ_{22}.

In the depicted simulation, the black and red arrows connect the configurations so that they correspond to the patterns depicted on the left side. Instead, it is also possible to "cross" the patterns: in this case, we either start with one pattern and end with the other or vice versa. Figure 4.8(a) shows two possibilities to cluster the configurations to patterns.

At the third state in the simulation, the arrows form a graph that is shown again in Figure 4.8(b). It has two inputs and two outputs. As all configurations contain the same state of $\mathcal{A}_\Delta$, all non-epsilon configurations can be succeeded by all epsilon configurations. When reconstructing a run of $\mathcal{A}$ out of a run of $\mathcal{B}$, is is not possible to distinguish between both possibilities. Both possibilities accept the same tree because in

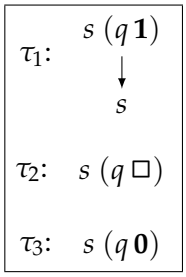


Figure 4.9: Transitions for Example 4.5

both possibilities the core nodes of the patterns together are the same. In the example, the leaf nodes $q_4\,0$ and the root node $q_1\,1$ can be either covered by the left or the right pattern. ▽

The previous example showed that ambiguous runs of $\mathcal{B}$ can be interpreted in multiple ways to reconstruct a run of $\mathcal{A}$. This can be generalized to all such situations. Definition 4.3 of a tiling requires that every node of the input tree is the core node of a partial subtree in the tiling. But in fact, being a core node of a partial subtree corresponds to being covered by a transition of $\mathcal{A}_\Delta$. The only requirement is that these transitions are clustered to patterns such that there is a successful run for every pattern. The example above shows that if there are multiple possibilities to cluster transitions, or configurations, to patterns, either possibility is either possibility leads to valid patterns.

4.2.1 The Simulating Root-to-Leaf Automaton

This subsection defines the simulating tree automaton formally. To facilitate the understanding of the definitions, the following example introduces some names first that will be used in the definition of the simulating automaton $\mathcal{B}$. It illustrates the construction of the automaton $\mathcal{B}$ for Example 4.3 formally.

Example 4.5. Recall that the regular tree automaton defined in Example 4.3 is: $\mathcal{A} = (Q, \Sigma, \mathcal{A}_\Delta, A)$ with $Q = \{q\}, \Sigma = \{\mathbf{1}, \mathbf{0}\}$ and $A = \{q\}$. The transition automaton is $\mathcal{A}_\Delta = (Q', Q \times (\Sigma \cup \{\square\}), I', \Delta', F')$ with $Q' = \{s\}$, $I' = \{s\}$ and $F' = \{\tau_2 = (s, (q, \square)), \tau_3 = (s, (q, \mathbf{0}))\}$. The only other transition $\tau_1 \in \Delta'$ together with the final transitions are illustrated in Figure 4.9. To differentiate better between the position $0 \in \mathbb{N}$ and the symbol $\mathbf{0} \in \Sigma$, we typeset the latter in bold font.

The simulating root-to-leaf automaton is $\mathcal{B} = (Q'', \Sigma, I'', \Delta'', F'')$. The set of positions occurring in transitions is $D = \{\epsilon, 0\}$. The set of $\mathcal{B}$-states is

$$Q'' = Q \times 2^{\{s\} \times \{\tau_1, \tau_2, \tau_3\} \times \{\epsilon, 0\}}.$$

$\texttt{initial\_configs}(q)$ is the set of all initial configurations that fit $\mathcal{A}$-state q:

$$\texttt{initial\_configs}(q) = \{(s, \tau_1, \epsilon), (s, \tau_2, \epsilon), (s, \tau_3, \epsilon)\}$$

Then the set I'' of initial states is

$$I'' = \{(q, \{\phi \mid \phi \neq \emptyset, \phi \subseteq \texttt{initial\_configs}(q)\})\}.$$

Next we are going to define the final assignment F'':

$$F''(\mathbf{0}) = \{(q, \varrho \cup \xi \cup \psi) \mid \varrho \neq \emptyset\}$$

The subsets of configurations ϱ, ξ and ψ are defined by the following abbreviations. $\texttt{final\_configs}(q, \sigma)$ consists of all final configurations fitting $\mathcal{A}$-state q and σ. $F''(\mathbf{0})$ must contain at least one final configuration fitting the symbol $\mathbf{0}$ (the condition $\varrho \neq \emptyset$ ensures this). As there is only one, this has to be contained:

$$\varrho \subseteq \texttt{final\_configs}(q, \mathbf{0}) = \{(s, \tau_3, \epsilon)\}$$

Furthermore, other patterns are allowed to be completed, but it is not required:

$$\xi \subseteq \texttt{final\_configs}(q, \square) = \{(s, \tau_2, \epsilon)\}$$

All final configurations belong to transitions that may be preceded by non-final configurations. $\texttt{prec}(\tau)$ consists of all non-epsilon configurations of transitions preceding τ:

$$\psi \subseteq \texttt{prec}(\tau_3) = \{(s, \tau_1, 0)\}$$

We can now fully define the set F'' of final transitions. F'' is only defined for symbols with arity zero, i.e., for symbol $\mathbf{0}$. As stated above, the subsets ξ and ψ may or may not contain configurations.

$$\begin{aligned} F''(\mathbf{0}) = \{&(q, \{(s, \tau_3, \epsilon)\}), \\ &(q, \{(s, \tau_3, \epsilon), (s, \tau_2, \epsilon)\}), \\ &(q, \{(s, \tau_3, \epsilon), (s, \tau_1, 0)\}), \\ &(q, \{(s, \tau_3, \epsilon), (s, \tau_2, \epsilon), (s, \tau_1, 0)\})\} \end{aligned}$$

Finally, we are going to introduce the transition relation Δ''. Its formal definition is

rather complicated. In this example, we omit some abbreviations that are only used to verify certain conditions and state the conditions intuitively instead. Transitions are only defined for symbols with non-zero arity, i.e., in this example only for the symbol **1**. Additionally, there is only one $\mathcal{A}$-state q. The conditions with respect to the transition are thus simplified to

$$\begin{aligned} &\big((q,\varphi),\mathbf{1},(q,\varphi_0)\big) \in \Delta'' \text{ iff} \\ &\text{all configurations in } \varphi \text{ fit } \mathcal{A}\text{-state } q, \text{ and symbol } \mathbf{1} \text{ or } \square \\ &\texttt{nonfinal\_}\epsilon\texttt{\_configs}(\varphi) \neq \emptyset \\ &\varphi_0 \in \texttt{configs}(\varphi, q, 0). \end{aligned} \tag{4.1}$$

The $\mathcal{B}$-state (q,φ) denotes the state at the current node and (q,φ_0) denotes the state at the 0th child.

The set $\texttt{nonfinal\_}\epsilon\texttt{\_configs}(\varphi)$ consists of all non-final epsilon configurations in φ. There is only one such configuration for $\mathcal{A}_\Delta$. It follows that this configuration is contained in $\texttt{nonfinal\_}\epsilon\texttt{\_configs}(\varphi)$ if it is contained in φ.

$$\texttt{nonfinal\_}\epsilon\texttt{\_configs}(\varphi) = \begin{cases} \{(s,\tau_1,\epsilon)\}, & \text{if } (s,\tau_1,\epsilon) \in \phi \\ \emptyset, & \text{otherwise} \end{cases}$$

For the above stated Condition (4.1) on transitions in Δ'', this means that (s,τ_1,ϵ) must be contained in φ because otherwise the state at the current node would be final.

It is left to define `configs`. This set contains all possible sets of configurations that can be contained in the $\mathcal{B}$-state at the child. `configs` depends on the set of configurations φ at the current node. For this example, `configs` can be simplified to

$$\begin{aligned} \texttt{configs}(\varphi,q,0) = \big\{ &\varrho \cup \xi \cup \psi \mid \varrho \neq \emptyset, \xi \neq \emptyset \\ &\varrho = \{\texttt{follow}(\tau_1,0)\}, \\ &\xi \text{ contains epsilon configurations of succeeding transitions,} \\ &\psi \subseteq \texttt{initial\_configs}(q)\big\}. \end{aligned} \tag{4.2}$$

The subset ξ will be discussed in depth later. First, we discuss ϱ and ψ: $\texttt{follow}(\tau,p)$ is the non-epsilon configuration of transition τ at position p.

$$\texttt{follow}(\tau_1,0) = (s,\tau_1,0)$$

Thus, ϱ ensures that the epsilon configuration at the current node is followed by the non-epsilon configuration of the same transition at the child's node.

`initial_configs` has been explained before. ψ allows new patterns to be started at the child's node.

Additionally, transitions in Δ'' must ensure that every non-final transition is succeeded by another transition. ξ handles this. Its formal definition is

$$\xi \in \left\{ P \subseteq \bigcup \texttt{all\_succ}(\varphi) \;\middle|\; \forall S \in \texttt{all\_succ}(\varphi).\, P \cap S \neq \emptyset \right\}. \tag{4.3}$$

First, we introduce the set $\texttt{succ}(\tau, p, q)$ which consists of all epsilon configurations of transition that can succeed τ at position p and that fit $\mathcal{A}$-state q. Only one transition has succeeding transitions for $\mathcal{A}_\Delta$ because the other transitions are final:

$$\texttt{succ}(\tau_1, 0, q) = \{(s, \tau_1, \epsilon), (s, \tau_2, \epsilon), (s, \tau_2, \epsilon)\}$$

For every non-final epsilon configuration, the set $\texttt{all\_succ}(\varphi)$ contains one set consisting of all succeeding transitions . As there is only one such configuration, `all_succ` only contains exactly one set:

$$\texttt{all\_succ}(\varphi) = \{\texttt{succ}(\tau_1, 0, p)\}$$

ξ ensures that every non-final transition must be succeeded by at least one successor but multiple successors are allowed. Thus, `all_succ` contains sets and of each of this sets, P must contain at least one element. Now, we can define the set of transitions.

$$\Delta'' = \left\{ \left((q, \{\overbrace{(s, \tau_1, \epsilon)}^{(4.1)}, \underbrace{(s, \tau_2, \epsilon)}_{\text{optional}}, \underbrace{(s, \tau_1, 0)}_{\text{optional}}\}), \mathbf{1}, (q, \{\overbrace{(s, \tau_1, 0)}^{\varrho}, \underbrace{\overbrace{(s, \tau_i, \epsilon)_{i=1,2,3}}^{\text{also part of } \psi}}_{\text{at least one, (4.3)}}\}) \right) \right\}$$

The two configurations annotated *optional* are allowed to be in the transition but are not necessary. So in fact, there are four possible choices for the state of the current node. Additionally, the three configurations annotated *at least one* also denote seven possibilities. Together, the set Δ'' contains $4 \cdot 7 = 28$ transitions.

As discussed before, the first pair, in front of the **1**, is the state of the current node, the second pair is the state of the child node.

The current node must always contain the configuration (s, τ_1, ϵ) as ensured by Condition 4.1. The child always contains $(s, \tau_1, 0)$ which is included in ϱ of the Definition (4.2) of `configs`. One of the three configurations $(s, \tau_i, \epsilon)_{i=1,2,3}$ must be contained in the

set of configurations at the child which is ensured by ξ of Equation (4.2) defined in (4.3). Finally, the set ψ does not provide more configurations because all configurations $(s, \tau_i, \epsilon)_{i=1,2,3}$ are also contained in ψ of Equation (4.2). ▽

The remainder of this subsection defines the *simulating tree automaton* formally and in general. Recall the setup: Let $\mathcal{A} = (Q, \Sigma, \mathcal{A}_\Delta, A)$ be a regular tree automaton with transition automaton $\mathcal{A}_\Delta = (Q', Q \times (\Sigma \cup \{\square\}), I', \Delta', F')$. The set of all positions in the internal transitions is $D = \{\epsilon\} \cup \{0, ..., k-1\}$ with $k = \max\{\text{arity}(\sigma) \mid \sigma \in \Sigma\}$. We shall construct a root-to-leaf automaton $\mathcal{B} = (Q'', \Sigma, I'', \Delta'', F'')$ such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{B})$.

The set of states has already been defined:

$$Q'' = Q \times 2^{Q' \times (\Delta' \cup F') \times D}$$

Let `Conf` be the set of configurations of $\mathcal{A}_\Delta$:

$$\texttt{Conf} = Q' \times (\Delta' \cup F') \times D$$

The set $\texttt{initial\_configs}(q)$ is the set of all initial configurations fitting $\mathcal{A}$-state q:

$$\texttt{initial\_configs}(q) = \{(s, \tau, \epsilon) \in \texttt{Conf} \mid s \in I', \tau(\epsilon) = (s, (q, \sigma))\}$$
$$\text{for } q \in Q$$

The initial states I'' of $\mathcal{B}$ depend on the accepting states A of $\mathcal{A}$ and on the initial states I' of $\mathcal{A}_\Delta$. At least one pattern has to start at every initial state of $\mathcal{B}$ but multiple patterns are possible.

$$I'' = \left\{(q, \phi) \mid q \in A, \phi \neq \emptyset, \phi \subseteq \texttt{initial\_configs}(q)\right\} \tag{4.4}$$

The final assignment F'' depends on F' of $\mathcal{A}_\Delta$ because for a run of $\mathcal{A}$ on a tree t, every pattern has to be completed. The definition uses two abbreviations `final_configs` and `prec`. $\texttt{final\_configs}(q, \sigma)$ is the set of final configurations fitting $\mathcal{A}$-state q and symbol σ:

$$\texttt{final\_configs}(q, \sigma) = \{(s, \tau, \epsilon) \in \texttt{Conf} \mid \tau \in F', \tau(\epsilon) = (s, (q, \sigma))\}$$
$$\text{for } q \in Q, \sigma \in \Sigma$$

Secondly, $\texttt{prec}(\tau)$ is the set of all non-epsilon configurations of transitions preceding τ:

$$\texttt{prec}(\tau') = \{(s, \tau, p) \in \texttt{Conf} \mid \tau \in \Delta', p \neq \epsilon, \tau'(\epsilon) = (s, (q, \sigma)), \tau(p) = s\}$$
$$\text{for } \tau' \in \Delta' \cup F'$$

F'' is a mapping defined for all symbols σ of arity zero. ϱ contains final configurations fitting σ. At least one such configuration must be contained in $F''(\sigma)$. ξ contains final configurations with transitions completing other patterns. Finally, ψ contains non-final and non-epsilon configurations of transitions preceding one of the former configurations.

$$\begin{aligned} F''(\sigma) = \big\{(q, \varrho \cup \psi \cup \xi) \mid q \in Q, \varrho \neq \emptyset,& \\ \varrho \subseteq \texttt{final\_configs}(q, \sigma),& \quad (4.5a) \\ \xi \subseteq \texttt{final\_configs}(q, \square),& \quad (4.5b) \\ \psi \subseteq \{\texttt{prec}(\tau) \mid \exists (s, \tau, \epsilon) \in (\varrho \cup \xi)\}\big\}& \quad (4.5c) \end{aligned}$$

Note that $F''(\sigma)$ contains invalid $\mathcal{B}$-states. For example, $\texttt{prec}(\tau)$ consists of non-epsilon configurations with all positions allowed by the arity. $F''(\sigma)$ is globally defined and not restricted to one position. The transition relation Δ'' ensures that the positions are valid in a run of $\mathcal{B}$.

The set of transitions Δ'' of $\mathcal{B}$ is built from transitions in Δ' of $\mathcal{A}_\Delta$. The following formula defines conditions that every transition in Δ'' must satisfy:

$$\begin{aligned} \big((q, \varphi), \sigma, (q_0, \varphi_0), ..., (q_{k-1}, \varphi_{k-1})\big) \in \Delta'' \text{ iff}& \\ \forall (s, \tau, \epsilon) \in \varphi : \tau(\epsilon) = (s, (q, \sigma)) \vee \tau(\epsilon) = (s, (q, \square)) \text{ and}& \quad (4.6a) \\ \texttt{nonfinal\_}\epsilon\texttt{\_configs}(\varphi) \neq \emptyset \text{ and}& \quad (4.6b) \\ \forall i : (q_i, \varphi_i) \in \{(q_i, P) \mid q_i \in Q, P \in \texttt{configs}(\varphi, q_i, i), \texttt{succ\_possible}(\varphi, q_i, i)\}& \quad (4.6c) \\ \text{for symbol } \sigma \in \Sigma \text{ with } \text{arity}(\sigma) = k > 0& \end{aligned}$$

(q, φ) is the $\mathcal{B}$-state of the current node and (q_i, φ_i) are the states at the children's nodes.

Condition (4.6a) ensures that the configurations at the current node fit the $\mathcal{A}$-state q. Additionally, the condition ensures that the configurations fit the symbol σ or the symbol $\square$. The set $\texttt{nonfinal\_}\epsilon\texttt{\_configs}(\varphi)$ returns all non-final epsilon configurations

included in φ:

$$\begin{aligned}&\texttt{nonfinal\_}\epsilon\texttt{\_configs}(\varphi) = \{(s,\tau,\epsilon) \in \varphi \mid \tau \in \Delta'\}\\&\text{for } \varphi \subseteq \texttt{Conf}\end{aligned}$$

Condition (4.6b) ensures that the set of configurations at the current node contains at least one epsilon configuration of a non-final transition. This is necessary, because otherwise the $\mathcal{B}$-state at the current node is final.

The last condition (4.6c) defines the $\mathcal{A}$-state and the configurations for the children of the current node. It uses the mappings `configs` and `succ_possible` that will be defined in the following. Intuitively, if `succ_possible` is false, any run of the automaton ends in a "dead end" at this node. $\texttt{configs}(\varphi, q_i, i) \subseteq 2^{\texttt{Conf}}$ consists of all possibilities to distribute configurations to the states of the children. One such possibility P is selected.

`succ_possible` uses the mappings `follow` and `succ`. Non-final epsilon configurations at the current node have to be followed at the children's nodes by a non-epsilon configuration with the same transition. The mapping `follow` defines this configuration:

$$\begin{aligned}&\texttt{follow}(\tau, i) = (\tau(i), \tau, i)\\&\text{for } \tau \in \Delta', i \in D\end{aligned} \tag{4.7}$$

The set $\texttt{succ}(\tau, p, q)$ consists of all epsilon configurations of succeeding transitions of τ at position p that fit $\mathcal{A}$-state q. It is dual to `prec`.

$$\begin{aligned}&\texttt{succ}(\tau, p, q) = \{(s, \tau', \epsilon) \in \texttt{Conf} \mid \tau(p) = s, \tau'(\epsilon) = (s, (q, \sigma))\}\\&\text{for } \tau \in \Delta', p \in D \setminus \{\epsilon\}, q \in Q\end{aligned} \tag{4.8}$$

`succ_possible` tests that there exist succeeding transitions. If there are no succeeding transitions available, then any run of $\mathcal{B}$ is in a "dead end" at the current node.

$$\begin{aligned}&\texttt{succ\_possible}(\varphi, q_i, i) = (\forall (s,\tau,\epsilon) \in \texttt{nonfinal\_}\epsilon\texttt{\_configs}(\varphi) : \texttt{succ}(\tau, i, q_i) \neq \emptyset)\\&\text{for } \varphi \subseteq \texttt{Conf}, q_i \in Q, i \in D \setminus \{\epsilon\}\end{aligned}$$

The following set $\texttt{all\_succ} \subseteq 2^{\texttt{Conf}}$ defines all successors per non-final epsilon configuration:

$$\begin{aligned}&\texttt{all\_succ}(\varphi) = \{\texttt{succ}(\tau, i, q_i) \mid (s,\tau,\epsilon) \in \texttt{nonfinal\_}\epsilon\texttt{\_configs}(\varphi)\}\\&\text{for } \varphi \subseteq \texttt{Conf}\end{aligned}$$

Now, we shall define $\texttt{configs}(\varphi, q_i, i) \subseteq 2^{\texttt{Conf}}$ which is the set of all possibilities to distribute configurations to the children's states:

$$\begin{aligned}
&\texttt{configs}(\varphi, q_i, i) = \\
&\quad \Big\{ \varrho \cup \xi \cup \psi \mid \\
&\qquad \varrho = \{\texttt{follow}(\tau, i) \mid (s, \tau, \epsilon) \in \texttt{nonfinal_}\epsilon\texttt{_configs}(\varphi)\}, && \text{(4.9a)} \\
&\qquad \xi \in \Big\{ P \subseteq \bigcup \texttt{all_succ}(\varphi) \,\Big|\, \forall S \in \texttt{all_succ}(\varphi).\, P \cap S \neq \emptyset \Big\}, && \text{(4.9b)} \\
&\qquad \psi \subseteq \texttt{initial_configs}(q_i) \Big\} && \text{(4.9c)} \\
&\text{for } \varphi \subseteq \texttt{Conf}, q_i \in Q, i \in D \setminus \{\epsilon\}
\end{aligned}$$

configs defines a set of possible sets of configurations. Every subset $\theta = \varrho \cup \xi \cup \psi$ of configurations is possible at the children's state if it follows three rules:

1. For every non-final epsilon configuration at the current node, exactly one epsilon configuration of the same transition must be contained in θ. This is handled by ϱ in (4.9a).

2. Every transition has to be succeeded by at least one transition. $\bigcup$ all_succ defines all successors possible. ξ defines a subset of all successors that contains at least one successor per non-final epsilon configuration at the current node (cf. (4.9b)).

3. Finally, ψ (cf. (4.9c)) can contain initial configurations to allow the start of new patterns.

This completes the construction.

4.2.2 Algorithm to Create Run of $\mathcal{A}$ Out of Run of $\mathcal{B}$

For the proof below, it is necessary to construct a run of a regular tree automaton $\mathcal{A}$ from a run of a simulating root-to-leaf automaton $\mathcal{B}$. This paragraph describes an algorithm for this. The idea is to annotate and color the run of $\mathcal{B}$ in the same way as shown in the Figures 4.2, 4.4 and 4.7. Before describing the algorithm in general, we shall illustrate by means of an example the general idea.

Example 4.6. Let $\mathcal{A} = (Q, \Sigma, \mathcal{A}_\Delta, A)$ be a regular tree automaton with transition automaton $\mathcal{A}_\Delta = (Q', Q \times (\Sigma \cup \{\square\}), I', \Delta', F')$ and let $\mathcal{B} = (Q'', \Sigma, I'', \Delta'', F'')$ be the corresponding simulating root-to-leaf automaton.

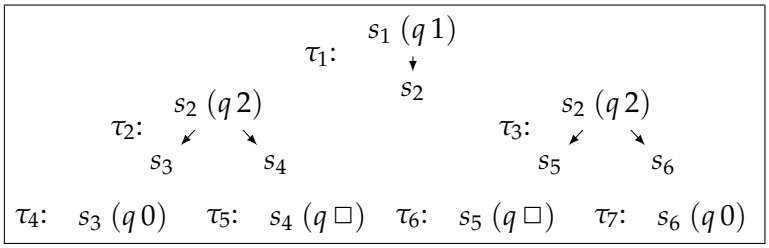


Figure 4.10: $\mathcal{A}_\Delta$'s transitions for Example 4.6

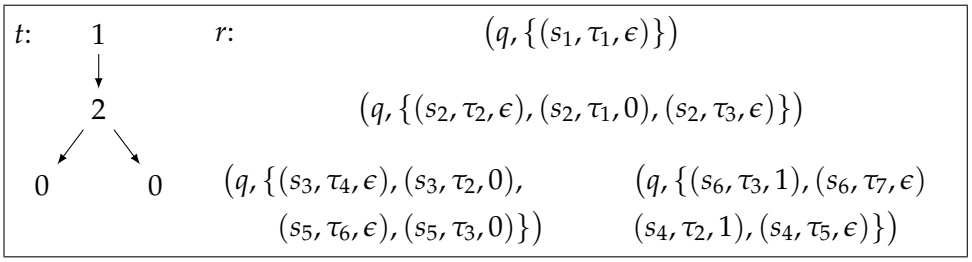


Figure 4.11: Run of $\mathcal{B}$ in Example 4.6 - Original run r

Let $Q = \{q\}$, $\Sigma = \{0, 1, 2\}$ and $A = \{q\}$. Furthermore, let $Q' = \{s_i\}_{1 \leq i \leq 6}$, $I' = s_1$ and $F' = \{\tau_i\}_{4 \leq i \leq 7}$. The set of transitions Δ' is shown in Figure 4.10.

Figure 4.11 shows a run r of $\mathcal{B}$ on a simple tree t. The accepted tree is shown on the left side and the run on the right. The states of $\mathcal{B}$ are shown in the same arrangement as the nodes of the accepted tree.

The goal is to reconstruct a run of $\mathcal{A}$ out of run r. For this, it is necessary to "cluster" the configurations of $\mathcal{A}_\Delta$ appearing in run r into patterns of $\mathcal{A}$. We do this by coloring the configurations that belong together in one pattern with the same color.

Let us first describe the general idea of the algorithm. The first three steps of the algorithm are only to visualize what we know already about the run. These visualizations are applied to Figure 4.12.

First, over- and underline initial and final configurations, respectively. Second, connect the epsilon configurations and the non-epsilon configurations of the same transitions by vertical arrows. Third, connect configurations part of the same $\mathcal{B}$-state that belong together because they are connected by the subformula `succ` by horizontal arrows. That means, if configuration γ' can possibly succeed configuration γ then connect γ with γ'.

The fourth step is to color the configurations. This may require backtracking. We start at the root of the tree with one color. Then we follow the arrows applied in step two and three to color the tree.

Figure 4.13 shows the first two configurations colored red. But now, the configuration

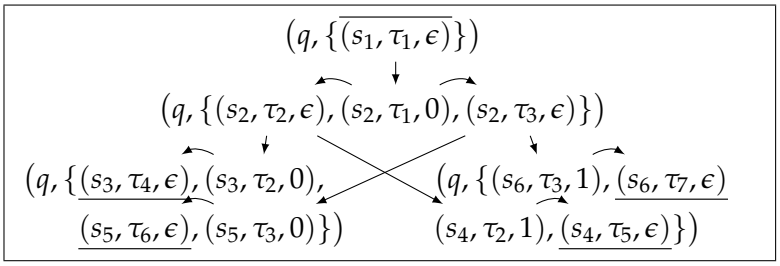


Figure 4.12: Run of $\mathcal{B}$ in Example 4.6 - First step

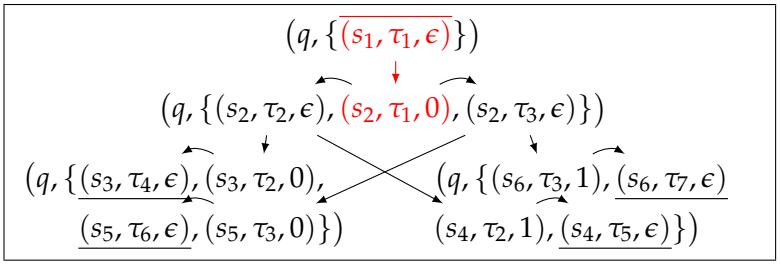


Figure 4.13: Run of $\mathcal{B}$ in Example 4.6 - Intermediate step

$(s_2, \tau_1, 0)$ has two successors. That means that from here on, intuitively two patterns are used in parallel in the run of $\mathcal{A}$. Both patterns need to have an initial configuration where the pattern begins. The beginning of the pattern that we colored red is the beginning of two patterns. It is therefore necessary to backtrack and recolor all red configurations in two colors at the same time. We choose the color blue in addition to red.

After that, the coloring continues without further issues as shown in Figure 4.14.

The first elements in the $\mathcal{B}$-states of the run express the values of $\mathcal{A}$'s run on t. Additionally, from the coloring of the run, it is possible to recreate the patterns used in the run of $\mathcal{A}$ as follows. The epsilon configurations in run r fit symbols and $\mathcal{A}$-states. Therefore, e.g., the configuration (s_1, τ_1, ϵ) for $\tau_1(\epsilon) = (s_1\,(q\,1))$ translates to $q\,1$ for the pattern. Translating all configurations this way and clustering them related to the colors returns the patterns shown in Figure 4.15. ▽

We need to describe the algorithm sketched above in more detail. Let $\mathcal{A}$ be a regular tree automaton, $\mathcal{B}$ a simulating root-to-leaf automaton, t a tree and r a run of $\mathcal{B}$ on t. We want to recreate a successful run $r_{\mathcal{A}}$ of $\mathcal{A}$ on t. As the $\mathcal{A}$-states are already contained in the $\mathcal{B}$-states of run r, only the patterns have to be reconstructed.

The algorithm works on a certain graph structure imposed on the configurations in the $\mathcal{B}$-states of r. In the following, this graph structures is defined. Every $\mathcal{B}$-state $r(p)$ for a position $p \in \mathrm{dom}(t)$ is of the form $(q, \{\gamma_1, ..., \gamma_k\})$. We denote by $r(p)_2$ the set of configurations at position p of run r. Let $G = (V, E)$ be a graph where V is the set of all

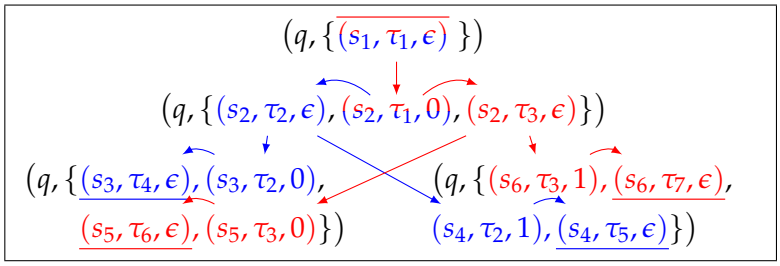


Figure 4.14: Run of $\mathcal{B}$ in Example 4.6 - Last step

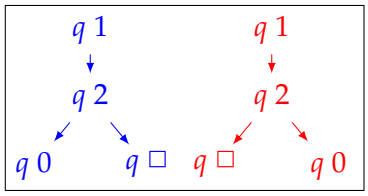


Figure 4.15: $\mathcal{A}$'s patterns for Example 4.6

configurations in r:

$$V = \bigcup_{p \in \mathrm{dom}(t)} \bigcup_{\gamma \in r(p)_2} \gamma$$

The set E of edges consists of two subsets: $E = E_1 \cup E_2$. The set of horizontal edges is defined as follows:

$$E_1 = \Big\{ \big((s, \tau, n), (s, \tau', \epsilon)\big) \in V^2 \;\Big|\; p \in \mathrm{dom}(t), n \in \mathbb{N}, (s, \tau, n) \in r(p)_2, (s, \tau', \epsilon) \in r(p)_2 \Big\} \quad (4.10a)$$

The set of vertical edges is defined as follows:

$$E_2 = \Big\{ \big((s, \tau, \epsilon), (s', \tau, n)\big) \in V^2 \;\Big|\; p \in \mathrm{dom}(t), n \in \mathbb{N}, (s, \tau, \epsilon) \in r(p)_2, (s', \tau, n) \in r(pn)_2 \Big\} \quad (4.10b)$$

Note that horizontal edges corresponds to the relation induced by `succ` defined in Equation (4.8) and vertical edges correspond to `follow` defined in Equation (4.7) of the construction of the simulating tree automaton.

Proposition 4.1. *There exist $\Gamma_1, \ldots, \Gamma_k \subseteq V$ such that $\bigcup_{1 \leq i \leq k} \Gamma_i = V$ and every Γ_i with respect to $E \cap \Gamma_i \times \Gamma_i$ is a tree of configurations where the configuration at the root is initial and the configurations at the leaves are final.*

In the examples above, the sets Γ_i correspond to all configurations in the same color.

Every set Γ_i represents one pattern used in the run $r_{\mathcal{A}}$ of $\mathcal{A}$. Thus, the proposition can be rephrased that out of a run r of $\mathcal{B}$, the successful runs of $\mathcal{A}_\Delta$ for all patterns of $\mathcal{A}$ in $r_{\mathcal{A}}$ can be reconstructed.

Proof. We prove the proposition by describing an algorithm to find the sets Γ_i. The algorithm is recursive. Every execution path handles one pattern or set Γ_i. Whenever a configuration has multiple successors or a new pattern is started, another execution path of the algorithm is started to handle the other pattern.

Every recursive step of the algorithm starts with an epsilon configuration and a set Γ_i and follows the graph for one transition of $\mathcal{B}$. It adds all configurations to the set Γ_i that belong to the same pattern. If the pattern is completed in this step, this execution path stops and outputs a set Γ_i, otherwise, the algorithm recurses with the succeeding epsilon configuration at the next node of tree t and the same set Γ_i.

The overall structure of the algorithm is as follows: We start an execution path for every configuration at the root of t together with empty sets Γ_i. Every recursion path will eventually stop and return a set Γ_i.

Algorithm: The $\mathcal{B}$-state at the root must be contained in I'' defined by Equation (4.4). The only configurations contained in states of I'' are initial configurations (s, τ, ϵ).

Let k be the amount of initial configuration $\gamma_1, ..., \gamma_k \in r(\epsilon)_2$ at the root. Let $m := k$ be a global counter indicating how many sets Γ_i are already in use. We start the procedure $\texttt{Step}(\gamma_i, \Gamma_i)$ for $1 \leq i \leq k$ and where Γ_i are empty sets.

Recursive procedure: $\texttt{Step}(\gamma = (s, \tau, \epsilon), \Gamma_i)$
Add γ to Γ_i.

Let $p \in \mathrm{dom}(t)$ be the position of this node in tree t. If $\tau \in F'$ then the pattern induced by Γ_i is completed at this node $t(p)$. Γ_i satisfies the condition of Proposition 4.1 because γ is final, the first configuration added to Γ_i was initial and all configurations are connected by the graph.

Otherwise, τ is a non-final transition. Depending on the arity, the node $t(p)$ has several children. As (4.6c) defines successors for every child, every child must contain a subtree of configurations belonging to Γ_i. All those configurations are added to the same set Γ_i, thus the next part of the procedure `Step` can be thought of as an iterative loop over all children.

Consider the iteration of the algorithm where the ith child of p is processed. Then the current node is $t(pi)$. Let Γ^* be the set of configurations at this node. Let q_i be the

$\mathcal{A}$-state at this node. It is possible to find q_i from one of the epsilon configurations in Γ^* because every $\mathcal{B}$-state contains at least one epsilon configuration.

As γ is non-final and an epsilon configuration, it follows that it must be contained in $\texttt{nonfinal\_}\epsilon\texttt{\_configs}(\Gamma^*)$ (cf. (4.6b)). In this case, there is a transition in $\mathcal{B}$ starting at the node with γ as configuration. Condition (4.6a) must be true, as otherwise the run r would not be successful. Condition (4.6c) consists of two parts: The first part is $\texttt{succ\_possible}(\Gamma^*, q_i, i)$, which is true. The second part verifies that the configurations in the child nodes are in $\texttt{configs}(\Gamma^*, q_i, i)$. This set contains several possibilities but every possibility consists of three subsets:

1. The first subset contains exactly one configuration, namely $\gamma' = (\tau(i), \tau, i)$ for the transition τ (cf. (4.9a)). The two configurations γ and γ' are connected by a vertical edge in E_2 (cf. (4.10b)). Add γ' to Γ_i.

2. The second subset (cf. (4.9b)) contains multiple configurations. There is at least one configuration succeeding every element in $\texttt{nonfinal\_}\epsilon\texttt{\_configs}(\Gamma^*)$.
   In particular, there is at least one configuration succeeding γ. Denote the configurations succeeding γ by $\gamma_1, ..., \gamma_k$. From the definition of succ, all configurations $\gamma_1, ..., \gamma_k$ have the same state as γ' and are therefore connected to γ' by a horizontal edge in E_1 (cf. (4.10a)).
   Call $\texttt{Step}(\gamma_i, \Gamma_{m+i})$ for $1 \leq i \leq k-1$ where $\Gamma_{m+i} = \Gamma_i$ and recurse by calling $\texttt{Step}(\gamma_k, \Gamma_i)$. Increase the global counter $m := m + k - 1$.
   Note that the other elements in $\texttt{nonfinal\_}\epsilon\texttt{\_configs}(\Gamma^*)$ are dealt with by another execution path and therefore also that path will deal with the other configurations succeeding those elements.

3. The third subset contains configurations of new patterns (cf. (4.9c)). For every such configuration γ^*, call $\texttt{Step}(\gamma^*, \Gamma_m)$ where Γ_m is an empty set and increase the global counter $m := m + 1$.

The above algorithm adds every $\gamma \in V$ to one of the subsets Γ_i because all initial configurations and all configurations created by the transition relation Δ'' are covered by the steps above. It is easy to see that this algorithm works. It follows that every Γ_i is a tree and can be interpreted as a pattern of $\mathcal{A}$. □

We can now state and prove the main result of this chapter.

Theorem 4.2. *The expressive power of regular tree automata equals the expressive power of finite root-to-leaf automata.*

Proof. There are two directions to show:

It can easily be seen that a finite root-to-leaf automaton $\mathcal{A}$ can be simulated by a regular tree automaton $\mathcal{B}$. As the transition relation of automaton $\mathcal{A}$ consists of finitely many transitions, these transitions can easily be accepted by some other tree automaton $\mathcal{B}'$. In this case, $\mathcal{B}$ having $\mathcal{B}'$ as transition automaton accepts the same language as $\mathcal{A}$.

For the other direction, we need to show that regular tree automata do not have more expressive power than root-to-leaf automata. We do so by simulating a regular tree automaton $\mathcal{A}$ by a finite root-to-leaf automaton $\mathcal{B}$ as discussed in Subsection 4.2.1.

It remains to be shown that $\mathcal{A}$ accepts the same language as $\mathcal{B}$. There are two directions to show.

$\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B})$:

Let t be a tree for which there exists a successful run r of $\mathcal{A}$. We construct a successful run of $\mathcal{B}$ for t. The first components on the $\mathcal{B}$-states are the states of r at the corresponding positions. Additionally, as the run r is successful, there is a tiling of $\mathcal{A}$ for t. This tiling can be partitioned into patterns accepted by $\mathcal{A}_\Delta$. The successful runs of $\mathcal{A}_\Delta$ on these patterns can be translated into the missing parts of the successful run of $\mathcal{B}$ on t. It can happen that the patterns of $\mathcal{A}$ overlap in a way that some $\mathcal{A}_\Delta$-states occur multiple times at the same node. As illustrated in Example 4.3, the resulting run of $\mathcal{B}$ is still successful.

$\mathcal{L}(\mathcal{B}) \subseteq \mathcal{L}(\mathcal{A})$:

Let t be a tree for which there exists a successful run r of $\mathcal{B}$. We construct a successful run of $\mathcal{A}$ for t. The run r contains a run of $\mathcal{A}$ in the first component of every state. To show that this run is successful, we need to find the successful runs of $\mathcal{A}_\Delta$ on all patterns.

The second components of the states in r contain configurations of $\mathcal{A}_\Delta$. The algorithm on in Proposition 4.1 clusters these configurations into patterns. As the configurations contain the transition, it is possible to recreate the runs of $\mathcal{A}_\Delta$ on these patterns. Thus, the run of $\mathcal{A}$ on t must also be successful. □

5 Conclusion and Outlook

Finite tree automata can be extended in multiple ways. Extensions to more complex input structures are one possibility. All these structures can be extended to its infinite counterpart. Infinite tree automata are one example. This work does not change the structure of the input but instead modifies the way how the input trees are processed.

Conventional finite tree automata use a transition relation that defines how the nodes of the input tree are labeled by states. The transition relation verifies local properties because it only accesses one node and its children. Thus, every transition is a tree of depth one.

Two extensions are proposed. The general finite tree automaton allows transitions of arbitrary size. They only need to be in tree shape. One major difference to conventional tree automata is that larger transitions can overlap. Nevertheless, general finite tree automata turn out to have the same expressive power as conventional finite tree automaton models from the literature. To prove this, the transitions of general finite tree automata are translated into transitions that do not overlap. The translation converts overlapping transitions into an upper and a lower part that are connected by a unique renaming. The non-overlapping transitions can easily be split into elementary transitions that technically correspond to the transitions of conventional finite tree automata.

The second extension changes the transition relation even further. Instead of using a finite set of transitions, the regular tree automaton uses a regular tree language as set of transitions. This implies that the amount of transitions can be infinite and the size of transitions can be unbounded. Although this automaton model seems to be more powerful, its expressive power is again equal to the conventional models from the literature. The proof simulates regular tree automata by conventional root-to-leaf automata. The states of the simulating automaton contain a set of configurations of

the transition automaton. The configurations correspond to the states of the transition automaton when it processes the patterns of the regular tree automaton.

Infinite trees were out of the scope of this work. But increasing the size of transitions and even using a regular set of transitions can also be done for tree automata working on infinite trees. One such automaton is the looping tree automaton whose only acceptance condition is that a run exists.

General finite tree automata can easily be extended to *general looping tree automata*. Everything can be adopted directly. Only a tiling is no longer a finite set of partial subtrees but an infinite set. The construction defined in Subsection 3.2.1 on page 19 only depends on the transitions. As the set of transitions stays finite even for infinite tree automata, the construction still works.

Unfortunately, the induction in Theorem 3.2 is based on the overlapping degree of a tiling, which is the sum of the overlapping degrees of all nodes in the tree t. In the infinite case, a tree t has infinitely many nodes. The overlapping degree of a tiling for an infinite tree can therefore be infinite. As a result, the proof provided for Theorem 3.2 does not work anymore in the infinite case.

To prove the correctness of the construction for infinite trees, a more local argument is necessary. As there are still only finitely many transitions, there are also only finitely many combinations and overlappings between them possible. This proof is left for future work.

Similarly to the general finite tree automaton, the regular tree automaton can also easily be adapted to infinite trees. The construction of a simulating tree automaton stays the same and even the proof for Lemma 4.2 should still work in the infinite case.

There is one challenge that is left open: the regular tree automaton defines its patterns by the transition automaton. Consequently, there might be arbitrary large patterns. In the finite case, this is no problem because the size of the input tree is a natural bound for every pattern. In the infinite case, it may happen that an infinite tree is tiled by one infinite pattern.

Bibliography

[AG68] M. A. Arbib and Y. Give'on. "Algebra automata I: Parallel Programming as a Prolegomena to the Categorical Approach". In: *Information and Control* 12.4 (1968), pp. 331–345.

[Baa05] F. Baader. *Automata and Logic*. TU Dresden. Lecture Notes. Mar. 2005. URL: `https://lat.inf.tu-dresden.de/teaching/ss2015/AL/aul.pdf`.

[BH67] M. Blum and C. Hewitt. "Automata on a 2-Dimensional Tape". In: *Switching and Automata Theory, 1967. SWAT 1967. IEEE Conference Record of the Eighth Annual Symposium on*. IEEE. 1967, pp. 155–160.

[Cou90] B. Courcelle. "The monadic second-order logic of graphs. I. Recognizable sets of finite graphs". In: *Information and computation* 85.1 (1990), pp. 12–75.

[Don70] J. Doner. "Tree Acceptors and Some of Their Applications". In: *Journal of Computer and System Sciences* 4 (1970), pp. 406–451.

[Fin15] B. Finkbeiner. *Automata, Games, and Verification*. Saarland University. Lecture Notes. 2015.

[Gia+96] D. Giammarresi, A. Restivo, S. Seibert, and W. Thomas. "Monadic Second-Order Logic Over Rectangular Pictures and Recognizability by Tiling Systems". In: *Information and Computation* 125.1 (1996), pp. 32–45.

[GR92] D. Giammarresi and A. Restivo. "Recognizable Picture Languages". In: *International Journal of Pattern Recognition and Artificial Intelligence* 6 (1992), pp. 241–256.

[GS15] F. Gécseg and M. Steinby. *Tree Automata*. 2015.

[LS97] M. Latteux and D. Simplot. "Recognizable Picture Languages and Domino Tiling". In: *Theoretical Computer Science* 178.1 (1997), pp. 275–283.

[Rab70] M. O. Rabin. "Weakly Definable Relations and Special Automata". In: *Mathematical Logic and Foundations of Set Theory*. Ed. by Y. Bar-Hillel. Vol. 59. 1970, pp. 1–23.

[Rab72] M. O. Rabin. "Automata of Infinite Objects and Church's Problem". In: *Regional Conference Series in Mathematics*. Vol. 13. American Mathematical Society, 1972, pp. 1–22.

[TATA] H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. *Tree Automata Techniques and Applications*. 2008.

[Tho90] W. Thomas. "Automata on Infinite Objects". In: *Handbook of Theoretical Computer Science, Volume B* (1990). Ed. by J. van Leeuwen, pp. 133–191.

[Tho91] W. Thomas. "On Logics, Tilings, and Automata". In: *Proceedings of the 18th International Colloquium on Automata, Languages and Programming*. Springer-Verlag. 1991, pp. 441–454.

[Tho97a] W. Thomas. "Automata Theory on Trees and Partial Orders". In: *Proceedings of the 7th International Joint Conference CAAP/FASE on Theory and Practice of Software Development*. TAPSOFT '97. London, UK: Springer-Verlag, 1997, pp. 20–38.

[Tho97b] W. Thomas. "Elements of an Automata Theory over Partial Orders". In: *Proceedings of the DIMACS Workshop on Partial Order Methods in Verification*. New York, NY, USA: AMS Press, Inc., 1997, pp. 25–40.

[TW68] J. W. Thatcher and J. B. Wright. "Generalized Finite Automata Theory with an Application to a Decision Problem of Second-Order Logic". In: *Mathematical Systems Theory* 2.1 (1968), pp. 57–81.

[VW86] M. Y. Vardi and P. Wolper. "Automata-Theoretic Techniques for Modal Logics of Programs". In: *Journal of Computer and System Sciences* 32 (1986), pp. 183–221.

Declaration of Authorship

I hereby confirm that this Master's thesis is my own work and that I have documented all sources and material used.

Dresden, September 13th, 2016

Signature (Sven Dziadek)