
ADVAINECD
OFRNAL LAAGGE
TNOHY THOHY

Advanced Formal Language Theory

Ryan Cotterell, Anej Svete, Alexandra Butoi, Andreas Opedal, Franz Nowak

Tuesday 11th July, 2023

Contents

Chapter 1

Tree Languages

1.1 Preliminaries

We first introduce some new terminology that will be useful later when working with finite-state tree automata.

Definition 1.1.1 (Arity)**.** *Let Σ be an alphabet of symbols. The **arity** of a symbol $a \in \Sigma$, which we will represent by $\mathsf{arity}(a)$, is a function $\mathsf{arity} : \Sigma \to \mathbb{N}$, representing the number of arguments the symbol can take.*

Definition 1.1.2 (Ranked alphabet)**.** *A **ranked alphabet** is a pair (Σ, arity) where Σ is a non-empty finite set of symbols (alphabet) and arity is the arity function of the symbols from Σ.*

In order to simplify the notation and mark more clearly the arity of the symbols, we will often overload Σ and use it for denoting both an alphabet and a ranked alphabet, like in the example below. We will mention explicitly which one we use, unless it is clear from the context.

Example 1.1.1. *Take for example the simple alphabet $\Sigma = \{a, b\}$. We will use the notation $\Sigma = \{a : 1, b : 0\}$ to denote a ranked alphabet over the set of symbols $\{a, b\}$, whose arities are $\mathsf{arity}(a) = 1$ and $\mathsf{arity}(b) = 0$.*

The set of symbols of arity p is denoted as $\Sigma^p \stackrel{\text{def}}{=} \{a \in \Sigma \mid \mathsf{arity}(a) = p\}$. Thus, the sets Σ^0, Σ^1, $\Sigma^2, \cdots \Sigma^p$ contain *constants*, *unary*, *binary*, $\cdots$, *p-ary* symbols and they are pairwise disjoint, i.e.

$$\forall i, j \in [0, n], \Sigma^i \cap \Sigma^j = \emptyset, n = \max(\{\mathsf{arity}(a) \mid a \in \Sigma\}). \tag{1.1}$$

In other words, each symbol can only have a single, fixed arity. You can think of the ranked alphabet as an extension of the alphabet that we saw in the context of finite-state automata, where we only allowed *unary* symbols.

Definition 1.1.3 (Ground Terms)**.** *The set $\mathcal{T}(\Sigma)$ of **ground terms**[1] defined of the ranked alphabet Σ is the smallest set defined inductively as follows:*

- $\Sigma^0 \subseteq \mathcal{T}(\Sigma)$,
- *if $n \geq 1$, $a \in \Sigma^n$ and $\boldsymbol{t}_1, \boldsymbol{t}_2, \cdots \boldsymbol{t}_n \in \mathcal{T}(\Sigma)$, then $a\langle \boldsymbol{t}_1, \boldsymbol{t}_2, \cdots \boldsymbol{t}_n \rangle \in \mathcal{T}(\Sigma)$.*

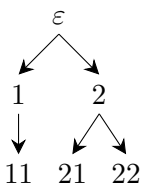
[1] *Ground terms* are a special type of *terms*, which do not contain variables. We chose not to introduce these as finite-state tree automata only work with ground terms.

Terms and trees We mentioned briefly that tree automata work with trees rather than strings, unlike the formalisms that we've seen previously. You may then wonder why we care about ground terms at all. Trees and terms are closely related, as we will see next, and we will use them interchangeably in the rest of this chapter.

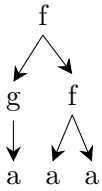
Definition 1.1.4 (Labeled tree)**.** *A **labeled tree** over a set of labels Σ is a mapping from a prefix-closed set $\mathcal{P}(\boldsymbol{t}) \subseteq \mathbb{N}^*$ onto Σ.*

A term $\boldsymbol{t} \in \mathcal{T}(\Sigma)$ can thus be visualized as a tree whose leaves are labeled with constant symbols and internal nodes are labeled with symbols of bounded arity. Each internal node has a number of children equal to its arity. The set $\mathcal{P}(\boldsymbol{t})$ is called a **tree domain** and it represents the set of positions (or nodes) in the tree. We will denote by $\boldsymbol{t}(p)$ the label at position $p \in \mathcal{P}(\boldsymbol{t})$ of the labeled tree $\boldsymbol{t}$.

Example 1.1.2. *Let $\mathcal{P}(\boldsymbol{t}) = \{\varepsilon, 1, 11, 2, 21, 22\}$ be a tree domain. Then the tree $\boldsymbol{t}$ has a topology as shown in Fig. 1.1a. Each node in the figure is labeled with its position. Consider further the ranked alphabet $\Sigma = \{f : 2, g : 1, a : 0\}$. The tree from Fig. 1.1b is a labeled tree over Σ.*



(a) Tree with domain $\mathcal{P}(\boldsymbol{t}) = \{\varepsilon, 1, 11, 2, 21, 22\}$ Each node in the tree is labeled with its position from $\mathcal{P}(\boldsymbol{t})$.



(b) Labeled tree with domain $\mathcal{P}(\boldsymbol{t}) = \{\varepsilon, 1, 11, 2, 21, 22\}$ over the ranked alphabet $\Sigma = \{f : 2, g : 1, a : 0\}$.

Figure 1.1: Example of labeled tree.

In the rest of the chapter we will simply call them *terms* and *trees*. This should not cause any confusion as we always mean ground terms or labeled trees. Now we presented the connection between trees and terms, we can look at some interesting examples.

Definition 1.1.5 (Subterm)**.** *The **subterm** of $\boldsymbol{t} \in \mathcal{T}(\Sigma)$ at position p, denoted by $\boldsymbol{t}|_p$, is the term defined by the following:*

- $\mathcal{P}(\boldsymbol{t}|_p) = \{j \mid pj \in \mathcal{P}(\boldsymbol{t})\}$,
- $\forall q \in \mathcal{P}(\boldsymbol{t}|_p), \boldsymbol{t}|_p(q) = \boldsymbol{t}(pq)$.

Example 1.1.3. *Consider the ranked alphabet $\Sigma = \{\wedge : 2, \vee : 2, \neg : 1, \mathsf{true} : 0, \mathsf{false} : 0\}$. Then the logical expressions $\wedge(\mathsf{true}, \neg(false))$ and $\wedge(\mathsf{true}, \vee(\mathsf{false}, \neg(\mathsf{false})))$ are ground terms from the set $\mathcal{T}(\Sigma)$. Their corresponding trees are shown in Fig. 1.2.*

Example 1.1.4. *Consider the alphabet $\Sigma = \{a, b\}$ and the special constant symbol $\langle \mathsf{EOS} \rangle$ that marks the end of a string. The string "abb" can be viewed as the term $t = a\langle b\langle b\langle \mathsf{EOS} \rangle\rangle\rangle$ over the ranked alphabet $\Sigma = \{a : 1, b : 1, \langle \mathsf{EOS} \rangle : 0\}$. Its corresponding tree is shown in Fig. 1.3.*

More generally, the alphabet that we saw in the context of finite-state automata is a ranked alphabet where we restrict the symbols to have arity 1 and strings over this alphabet can be interpreted as unary terms. We can thus think of the languages accepted by finite-state automata as sets of unary terms or chain-like trees.

Figure 1.2: Examples of ground terms.

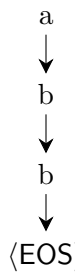


Figure 1.3: Visual representation of the term $a\,\langle b\,\langle b\,\langle \mathsf{EOS}\rangle\rangle\rangle$ as a tree.

A useful function of a tree is its height. As we use it extensively in proofs for finite-state tree automata, we define it below.

Definition 1.1.6 (Tree height)**.** *The* ***height*** *of a tree* $\boldsymbol{t}$*, denoted as* $\mathsf{height}(\boldsymbol{t})$*, is defined inductively by:*

- $\mathsf{height}(\boldsymbol{t}) = 1$ *if* $\boldsymbol{t} \in \Sigma^0$*,*
- $\mathsf{height}(\boldsymbol{t}) = 1 + \max(\{\mathsf{height}(\boldsymbol{t}_i) \mid i \in \{1, 2 \cdots, p\}\}$*, where* $\boldsymbol{t} = a\,\langle \boldsymbol{t}_1, \boldsymbol{t}_2, \cdots, \boldsymbol{t}_p\rangle$ *for some symbol* $a \in \Sigma^p$*.*

Example 1.1.5. *The heights of the trees from Fig. 1.2 are 3 and 4, respectively.*

1.2 Bottom-Up Finite-State Tree Automata

In the previous chapters we've seen formalisms that work with string inputs and outputs. However, many problems can be modeled naturally using more structured objects such as trees or graphs. For instance, arithmetic operations or logical formulas may be represented as strings but their semantics are best understood when looking at the parse tree of the expression. This motivates the introduction of another type of computational device called **finite-state tree automaton**. As we will see in detail later in this chapter, the **finite-state tree automaton** lies at the intersection of **finite-state automata** and **context-free grammars**, sharing many of their properties, but deals with trees rather than strings.

Finite-state tree automata are computational devices that determine whether trees are elements of a given **tree language**. Similar to string languages, there is also a hierarchy of tree languages. The class of tree languages that finite-state tree automata can recognize is the **class of regular tree languages**[2].

Definition 1.2.1 (Tree Language)**.** *A **tree language** L over the ranked alphabet (Σ, arity) is a subset of $\mathcal{T}(\Sigma)$.*

If this subset is the empty set, we call the language L **empty**; otherwise, we call it **non-empty**. Additionally, we call L **infinite** if it is an infinite set.

Definition 1.2.2 (Regular Tree Language)**.** *A tree language is **regular** if and only if it is accepted by a finite-state tree automaton.*

Finite-state tree automata come in two flavours when looking at the order in which they perform computation: **bottom-up** and **top-down**. As the name suggests, a bottom-up tree automaton starts its computation at the leaves of the tree and moves upwards until it reaches the root. A top-down tree automaton performs computation in reverse order, starting at the root of the tree and moving downwards until it reaches the leaves. We will start by defining formally the **bottom-up finite-state tree automaton** and present a variety of algorithms before moving to its top-down counterpart and making a comparison between the two.

Definition 1.2.3 (Bottom-Up Finite-State Tree Automaton)**.** *A **bottom-up finite-state tree automaton** (bottom-up FSTA) is a 5-tuple $(\Sigma, Q, I, F, \delta)$ where:*

- *Q is a finite set of states;*
- *Σ is a ranked alphabet;*
- *$I \subseteq Q$ is a set of initial states;*
- *$F \subseteq Q$ is a set of final or accepting states;*
- *$\delta \subseteq Q^n \times (\Sigma \cup \{\varepsilon\}) \times Q$ is a finite multi-set, $n = \max\{arity(a) \mid a \in \Sigma\}$ and elements of δ are generally called **transitions**.*

[2] Just like in the case of string languages, not all tree languages are regular. We can come up with examples of tree languages that are not accepted by any FSTA. Consider for instance the language $L = \{A(t, t) \mid t \in T(\Sigma)\}$, which consists of trees whose root is labeled with A and the left and right subtrees are identical. The language L belongs to the class of context-free tree languages.

A bottom-up finite-state tree automaton is *run* on ground terms over Σ. Before defining formally what a **run** on a tree is, let's give an informal description of how a bottom-up finite-state tree automaton works. The automaton starts at its leaves simultaneously and moves upwards, associating a state from Q to each node inductively. The states of the leaves are solely determined by their label. At each step, depending on the states of the children and the label of the node, a transition from δ determines the state of the node. A term (tree) is **accepted** if at the end of this process its root is associated with a state from F.

Initial States and Arity-0 Symbols. Traditionally,[3] bottom-up finite-state tree automata have no initial states and instead have "initial rules" of the form $a \to q$, where $a \in \Sigma^0$ is the label of a leaf and $q \in Q$. These initial rules would correspond to FSA-like transitions $\xrightarrow{a} q$ where no state is required on the left-hand side of the transition. Intuitively, this means that no state is required in order to end up in leaf (as it has no children) and its state is determined only by its label. However, the notation that we use in these notes differs slightly, allowing us to keep the notation consistent across different formalisms and emphasize better the similarities between finite-state tree automata and finite-state automata. In order to allow initial states, we introduce our own "initial rules" of the form $q_I \xrightarrow{a} q$, where the left-hand side of the transition must be an initial state $q_I \in I$. Additionally, we require that no transition from δ can have an initial state as target state, i.e. $\bullet \xrightarrow{\bullet} q_I, q_I \in I$. In this notation, the transitions for constant symbols become similar to those of the unary symbols.

Definition 1.2.4 (Run)**.** *A **run** of a finite-state tree automaton $\mathcal{A} = (\Sigma, Q, I, F, \delta)$ on a tree $\boldsymbol{t}$ is a mapping $r : \mathcal{P}(\boldsymbol{t}) \mapsto Q$, i.e. a labeling of $\boldsymbol{t}$'s nodes with states from Q using transition rules from δ.*

For every position $p \in \mathcal{P}(\boldsymbol{t})$ such that $\boldsymbol{t}(p) = a \in \Sigma^k$, $r(p) = q'$ if and only if $r(pi) = q_i, \forall i \in [1, k]$ and there is a transition $\langle q_1, \cdots, q_k \rangle \xrightarrow{a} q' \in \delta$.

Definition 1.2.5 (Accepting Run)**.** *A run of a finite-state tree automaton $\mathcal{A} = (\Sigma, Q, I, F, \delta)$ on a tree $\boldsymbol{t}$ is **accepting** if $\boldsymbol{t}$'s root gets labeled with a state from F.*

When talking about trees $\boldsymbol{t}$ for which there is an accepting run, we will often say that the term $\boldsymbol{t}$ is **accepted** or **recognized** by the automaton. Note that there might be multiple runs on a tree using transition rules from δ. In order for a finite-state tree automaton to accept a tree, we only require the existence of one accepting run.

Example 1.2.1. *Consider the tree automaton $\mathcal{A} = (\Sigma, Q, I, F, \delta)$ defined formally with*

- $Q = \{q_I, q_0, q_1\}$;
- $\Sigma = \{\wedge : 2, \vee : 2, \neg : 1, \mathit{true} : 0, \mathit{false} : 0\}$;
- $I = \{q_I\}$;
- $F = \{q_1\}$;
- $\delta = \{(\langle q_I \rangle, false, q_0), (\langle q_I \rangle, true, q_1), (\langle q_0 \rangle, \neg, q_1), (\langle q_1 \rangle, \neg, q_0), (\langle q_0, q_0 \rangle, \wedge, q_0), (\langle q_0, q_1 \rangle, \wedge, q_0), (\langle q_1, q_0 \rangle, \wedge, q_0), (\langle q_1, q_1 \rangle, \wedge, q_1), (\langle q_0, q_0 \rangle, \vee, q_0), (\langle q_0, q_1 \rangle, \vee, q_1), (\langle q_1, q_0 \rangle, \vee, q_1), (\langle q_1, q_1 \rangle, \vee, q_1)\}$.

[3] This is the notation used by a large proportion of the tree automata literature, including the classic textbook Comon et al. (2008).

Ground terms over Σ are boolean formulas without variables and a run over such ground term can be viewed as an evaluation of the corresponding boolean formula. Consider for example the ground term $\wedge(\vee(true, false), \neg(false))$ and its corresponding tree, shown in Fig. 1.4. As there exists a run in which the root is associated with a state from F (the run from Fig. 1.4 is accepting*), the term is recognized by $\mathcal{A}$. More generally, the tree language recognized by the tree automaton $\mathcal{A}$ is the set of boolean expressions that evaluate to true.*

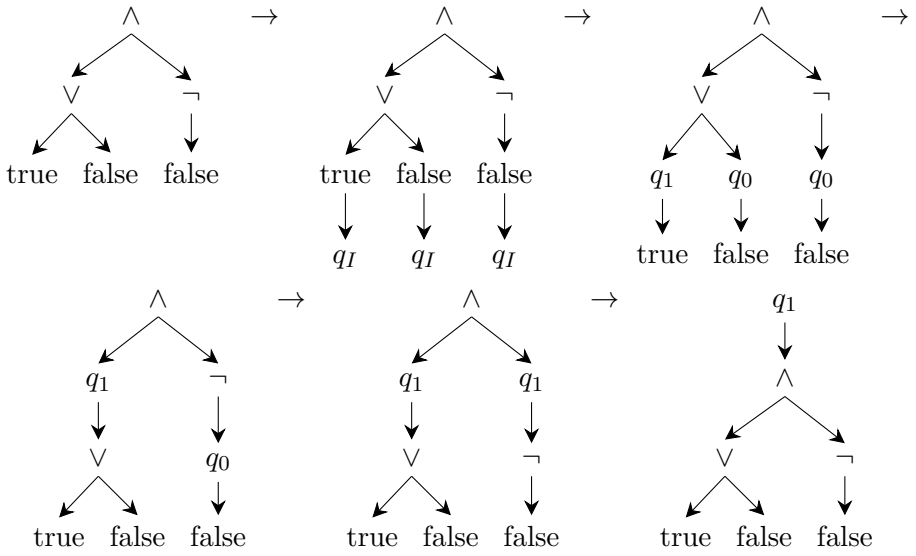


Figure 1.4: Example of a run on the ground term $\wedge(\vee(false, false), \neg(false))$.

Another view on FSTAs. In addition to the tree representation, finite-state tree automata can be also visually represented as hypergraphs. The nodes of the hypergraph are the set of states Q and the transitions are drawn as arcs with mutiple ordered source nodes (hyperedges), as dictated by the arity of the symbols. For this reason, we will often use the more intuitive notation $\langle q_1, q_2, \ldots, q_k \rangle \xrightarrow{a} q$ for a hyperedge when talking about transitions $(\langle q_1, q_2, \ldots, q_k \rangle, a, q) \in \delta$. The vector $\langle q_1, q_2, \ldots, q_k \rangle$ represents the ordered set of source nodes of the transition, the symbols on the hyperedges are the symbols from the ranked alphabet Σ and q is the target node of the transition. Therefore, a run of an FSTA on a tree can be seen as a hyperpath through the hypergraph, starting from the initial states. A hyperpath is a series of hyperedges that can be followed, sometimes in parralel, through the hypergraph. Further, a run is accepting if the final node of the hyperpath is a final state $q \in F$. When visualizing an FSTA as a hypergraph, we indicate the order of the source states on the edges unless it is obvious from the context.

Example 1.2.2. *Consider an algebra over the natural numbers in which the only possible operations are* succ*, which is a unary operation returning the successor of a number, and $+$, which performs addition of two numbers. Additionally, we treat 0 as a constant symbol. The set of expressions evaluating to even numbers can be viewed as a tree language and one finite-state tree automaton accepting this language can be formally defined as follows:*

- $Q = \{q_i, q_e, q_o\}$;
- $\Sigma = \{0 : 0, succ : 1, + : 2\}$;
- $I = \{q_i\}$;

- $F = \{q_e\}$;
- $\delta = \{(\langle q_i \rangle, 0, q_e), (\langle q_e \rangle, succ, q_o), (\langle q_o \rangle, succ, q_e), (\langle q_o, q_o \rangle, +, q_e), (\langle q_o, q_e \rangle, +, q_o), (\langle q_e, q_o \rangle, +, q_o), (\langle q_e, q_e \rangle, +, q_e)\}$

This automaton can be visualized as a hypergraph as shown in Fig. 1.5. We did not mark the order of the nodes in the left-hand side of the transitions $\langle q_o, q_o \rangle \xrightarrow{+} q_e$ *and* $\langle q_e, q_e \rangle \xrightarrow{+} q_e$ *as they are symmetric. Additionally, we collapsed the transitions* $\langle q_e, q_o \rangle \xrightarrow{+} q_o$ *and* $\langle q_o, q_e \rangle \xrightarrow{+} q_o$ *into a single one as they both lead to state* q_o.

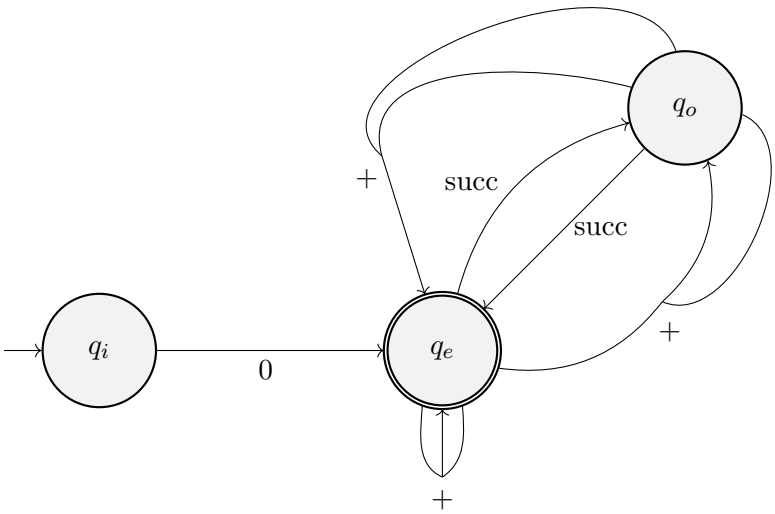


Figure 1.5: Example of FSTA visualized as a hypergraph.

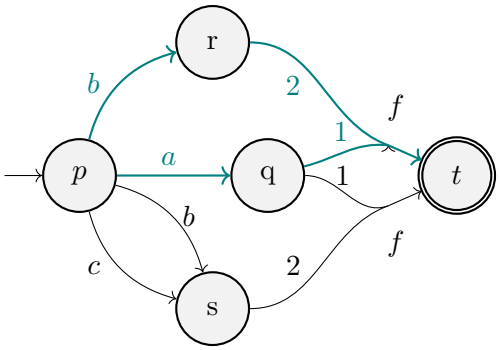
Definition 1.2.6 (Hyperpath)**.** *A* ***hyperpath*** $\boldsymbol{\pi}$ *is an element of the set of trees* $\mathcal{T}_\Sigma(Q)$*, where we define the set* $\mathcal{T}_\Sigma(Q)$ *inductively as follows:*

- $Q \subseteq \mathcal{T}_\Sigma(Q)$ *where* Q *is the set of states of the automaton,*
- $\langle \boldsymbol{\pi}_1, \cdots, \boldsymbol{\pi}_k \rangle \xrightarrow{a} q \in \mathcal{T}_\Sigma(Q)$ *if* $q \in Q$*,* $a \in \Sigma^k$*,* $\boldsymbol{\pi}_1, \cdots, \boldsymbol{\pi}_k \in \mathcal{T}_\Sigma(Q)$ *and there exists a transition* $\langle n(\boldsymbol{\pi}_1), \cdots, n(\boldsymbol{\pi}_k) \rangle \xrightarrow{a} q \in \delta$*,*

where $n(\boldsymbol{\pi})$ *denotes the state at the root of the hyperpath* $\boldsymbol{\pi}$*.*

In addition to $n(\boldsymbol{\pi})$, we use $p(\boldsymbol{\pi})$ to denote the set of states at the leaves of $\boldsymbol{\pi}$ and $\Pi(\mathcal{Q}, \mathcal{Q}')$ to denote the set of paths starting from states in $\mathcal{Q}$ and ending in a state from $\mathcal{Q}'$. Intuitively, a hyperpath is a series of transitions $\left(\boldsymbol{\pi} = \langle q_1, \cdots, q_k \rangle \xrightarrow{\bullet} q, \cdots, \langle q'_1, \cdots, q'_p \rangle \xrightarrow{\bullet} q'\right)$, where the states $q_1, \cdots, q_k, q, q'_1, \cdots, q'_p, q'$ must be sorted in topological order. Note that the topological order is not necessarily unique. By following hyperpaths starting from the initial states, we can generate trees from the set $\mathcal{T}(\Sigma)$. We call such a tree the **yield** of the hyperpath. Fig. 1.5 shows such an example.

So far we haven't imposed any restrictions on the set of transitions δ, allowing the existence of one, multiple or no transitions that process some symbol a from the source nodes $\langle q_1, q_2, \cdots, q_k \rangle$. Deterministic and complete FSTAs are special types of FSTAs that require the set of transitions to have certain properties.



(a) Example of a hyperpath through an FSTA.

(b) Yield of the hyperpath.

Figure 1.6: Example of hyperpath (highlighted in blue) through an FSTA and its yield.

Definition 1.2.7 (Deterministic FSTA)**.** *A finite-state tree automaton* $\mathcal{A} = (\Sigma, Q, I, F, \delta)$ *is called* ***deterministic*** *or* ***sequential*** *if for every* $(\langle q_1, q_2, \dots q_p \rangle, a) \in Q^p \times \Sigma$*,* $\mathsf{arity}(a) = p$*, there exists at most one state* $q \in Q$ *such that* $(\langle q_1, q_2, \dots q_p \rangle, a, q) \in \delta$ *and there is only one initial state i.e.,* $|I| = 1$*. Otherwise, we call* $\mathcal{A}$ ***non-deterministic****.*

Just like in the string case, determinism and ambiguity are two closely related concepts. In an **unambiguous** FSTA, there is at most one accepting run for any tree $\boldsymbol{t} \in \mathcal{T}(\Sigma)$. Otherwise, the FSTA is called **ambiguous**. While every deterministic FSTA is unambiguous, there exist unambiguous FSTAs that are non-deterministic.

Definition 1.2.8 (Complete FSTA)**.** *A finite-state tree automaton* $\mathcal{A} = (\Sigma, Q, I, F, \delta)$ *is called* ***complete*** *if for every* $(\langle q_1, q_2, \cdots q_p \rangle, a) \in Q^p \times \Sigma$*,* $\mathsf{arity}(a) = p$*, there exists at least one state* $q \in Q$ *such that* $(\langle q_1, q_2, \cdots q_p \rangle, a, q) \in \delta$*.*

Note that in a deterministic and complete automaton, there is *exactly one* transition $(\langle q_1, q_2, \cdots q_p \rangle, a, q) \in \delta$ for every $(\langle q_1, q_2, \cdots q_p \rangle, a) \in Q^p \times \Sigma$. This further implies that there is exactly one run for each tree $\boldsymbol{t} \in \mathcal{T}(\Sigma)$.

1.3 The Pumping Lemma

Before presenting the intuition behind the pumping lemma for regular tree languages, we need to present a new concept that will come up repeatedly, both in this section and later, in the context of minimization and weight pushing of tree automata.

Definition 1.3.1 (Context)**.** *The set $\mathcal{C}(\Sigma)$ of **contexts** defined over the ranked alphabet Σ is the smallest set defined inductively as follows:*

- $\Box \subseteq \mathcal{C}(\Sigma)$ *where $\Box$ is a special symbol that does not belong to the alphabet,*
- *If $\mathbf{c} \in \mathcal{C}(\Sigma)$, $\boldsymbol{t}_1, \boldsymbol{t}_2, \ldots, \boldsymbol{t}_{i-1}, \boldsymbol{t}_{i+1}, \ldots, \boldsymbol{t}_k \in \mathcal{T}(\Sigma)$ for $i = 1, \cdots, k$ and $a \in \Sigma^k$, then $a\langle \boldsymbol{t}_1, \boldsymbol{t}_2, \ldots, \boldsymbol{t}_{i-1}, \mathbf{c}, \boldsymbol{t}_{i+1}, \ldots, \boldsymbol{t}_k \rangle \in \mathcal{C}(\Sigma)$.*[4]

In words, a context is a tree with a gap (denoted with the special symbol $\Box$) that occurs only once a leaf position in the tree, as shown in Example 1.3.1. Notice that $\mathcal{C}(\Sigma) \cap \mathcal{T}(\Sigma) = \emptyset$, but $\mathcal{C}(\Sigma) \subseteq \mathcal{T}(\Sigma \cup \{\Box\})$. We denote by $\mathbf{c}[\boldsymbol{t}]$ the tree we obtain by substituting the symbol $\Box$ with the tree $\boldsymbol{t}$ in the context $\mathbf{c} \in \mathcal{C}(\Sigma)$. When $\boldsymbol{t} = \mathbf{c}'$ is also a context $\in \mathcal{C}(\Sigma)$, the resulting tree $\mathbf{c}'' = \mathbf{c}[\mathbf{c}']$ is also a context, as shown in Fig. 1.7d. We denote by $\mathbf{c}^n$ the successive substitution of $\mathbf{c}$ in $\mathbf{c}^i$, where $i \leq k$. It holds that $\mathbf{c}^0 = \Box$, $\mathbf{c}^1 = \mathbf{c}$ and for $i = 2, \cdots, n$,

$$\mathbf{c}^n = \mathbf{c}^{n-1}[\mathbf{c}] = \mathbf{c}^{n-2}[\mathbf{c}[\mathbf{c}]] = \cdots = \mathbf{c}[\underbrace{\cdots}_{n \text{ times}}[\mathbf{c}]]. \tag{1.2}$$

Furthermore, we call $\Box$ the **trivial context**.

Example 1.3.1. *Consider the ranked alphabet $\Sigma = \{f : 2, g : 1, a : 0, b : 0\}$. The tree from Figure 1.7a is an example of a context $\mathbf{c} \in \mathcal{C}(\Sigma)$. If we fill its gap $\Box$ with the tree from Figure 1.7b, we obtain the tree from Figure 1.7c.*

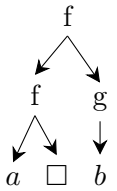
In the case of *string* languages, we can use the pumping lemma to show that a language is non-regular. Similarly, we can use the pumping lemma for tree languages to show that a set of trees is not regular. In addition to that, the corollary of the pumping lemma is useful for checking whether a tree language is empty or infinite. We start with an example that gives a high-level intuition behind the pumping lemma for tree languages.

Example 1.3.2. *This example is adapted from Comon et al. (2008). Consider the ranked alphabet $\Sigma = \{f : 2, g : 1, a : 0\}$ and the tree language $L = \{f(g^i(a), g^i(a)) \mid i > 0\}$. Now assume that L is recognized by some FSTA $\mathcal{A}$ with k states. Therefore, L must be regular. Consider the tree $\boldsymbol{t} = f(g^k(a), g^k(a))$ that belongs to L, thus there must exist at least one accepting run r on $\boldsymbol{t}$. As k is the cardinality of the set of states Q, there must exist two distinct positions on the left branch (we assume the left branch without loss of generality) of $\boldsymbol{t}$ that get labeled with the same state by r. Therefore, we could "cut" this part of the branch between the two positions, which would result in a term $\boldsymbol{t}' = f(g^j(a), g^k(a))$ for $j < k$ such that there is an accepting run on $\boldsymbol{t}'$. But $\boldsymbol{t}'$ is not in L, which leads to a contradiction.*

Lemma 1.3.1 (Pumping Lemma)**.** *Let L be a regular tree language. Then, there exists a constant $k > 0$ such that, for every term $\boldsymbol{t} \in L$ with $\mathsf{height}(\boldsymbol{t}) > k$, there exists a context $\mathbf{c} \in \mathcal{C}(\Sigma)$, a non-trivial context $\mathbf{c}' \in \mathcal{C}(\Sigma)$ and a term $\boldsymbol{t}' \in \mathcal{T}(\Sigma)$ such that $\boldsymbol{t} = \mathbf{c}[\mathbf{c}'[\boldsymbol{t}']]$, and, for all $n \geq 0$, $\mathbf{c}[\mathbf{c}'^n[u]] \in L$.*

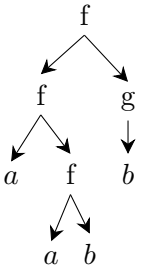
[4] Recall that Σ^k is defined as the set of symbols of arity k.

(a) An example of a context $\mathbf{c} \in \mathcal{C}(\Sigma)$.



(b) An example of a tree $\boldsymbol{t} \in \mathcal{T}(\Sigma)$.

(c) The tree $\mathbf{c}[\boldsymbol{t}]$ resulting from substituting $\boldsymbol{t}$ in C.



(d) The context $\mathbf{c}^2 = \mathbf{c}[\mathbf{c}]$ resulting from substituting the context $\mathbf{c}$ in $\mathbf{c}$.

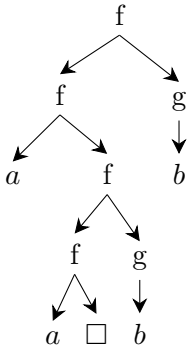


Figure 1.7: An example of a context $\mathbf{c}$ over the alphabet Σ, together with a term $\boldsymbol{t}$ over Σ and substitutions of a tree and a context in a context.

Proof. Let $\mathcal{A} = (\Sigma, Q, I, F, \delta)$ be an FSTA and $L = L(\mathcal{A})$ be $\mathcal{A}$'s language. Additionally, let $k = |Q|$, i.e. the number of states of $\mathcal{A}$. Consider a term $\boldsymbol{t} \in L$ with $\mathsf{height}(\boldsymbol{t}) > k$ and an accepting run r of $\mathcal{A}$ on $\boldsymbol{t}$. Now consider a path in $\boldsymbol{t}$, from a leaf to the root, of height greater than k. Notice that this is always possible as $\boldsymbol{t}$'s height is greater than k. As k is the cardinality of Q, by the pigeonhole principle, it follows that there must be two positions along this path, p_1 and p_2, such that such that $r(p_1) = r(p_2) = q$ for some state q. We can assume without loss of generality that $p_1 < p_2$. Let $\boldsymbol{t}'$ be the subtree of $\boldsymbol{t}$ at position p_2, i.e. $\boldsymbol{t}' = \boldsymbol{t}|_{p_2}$, and $\boldsymbol{t}''$ the subtree at position p_1, i.e. $\boldsymbol{t}'' = \boldsymbol{t}|_{p_1}$. Then there exists a non-trivial context $\mathbf{c}'$ such that $\boldsymbol{t}'' = \mathbf{c}'[\boldsymbol{t}']$ and a context $\mathbf{c}$ such that $\boldsymbol{t} = \mathbf{c}[\mathbf{c}'[\boldsymbol{t}]]$. Consider a tree $\mathbf{c}[\mathbf{c}'^n[\boldsymbol{t}'']]$ for some $n \geq 0$. Then there must exist an accepting run on this tree. ■

Example 1.3.3. *Let $\Sigma = \{a : 0, f : 2\}$ be a ranked alphabet and $L = \{\boldsymbol{t} \in \mathcal{T}(\Sigma) \mid |\mathcal{P}(\boldsymbol{t})| \text{ is a prime number}\}$. We can prove that L is not regular using the pumping lemma. Consider the tree $\boldsymbol{t} = f(f(a, f(a, a)), a)$ of height 4 with $|\mathcal{P}(\boldsymbol{t})| = 7$. For k equal to, say, 3, there exist contexts $\mathbf{c} = f(f(a, \square), a)$ and $\mathbf{c}' = f(\square, a)$, and the subterm $\boldsymbol{t}' = a$ such that $\boldsymbol{t} = \mathbf{c}[\mathbf{c}'[\boldsymbol{t}']]$. Then for $n = 2$, the term $\boldsymbol{t}'' = \mathbf{c}[\mathbf{c}'^2[\boldsymbol{t}']] = f(f(a, f(f(a, a), a)), a)$ has $|\mathcal{P}(\boldsymbol{t}'')| = 9$, thus $\boldsymbol{t}''$ is not in L.*

Corollary 1.3.1. *Let $\mathcal{A} = (\Sigma, Q, I, F, \delta)$ be an FSTA. Then $L = L(\mathcal{A})$ is non-empty if and only if there exists a term $\boldsymbol{t}$ in L with $\mathsf{height}(\boldsymbol{t}) \leq |Q|$. Additionally, L is infinite if and only if there exists a term $\boldsymbol{t}$ such that $|Q| < \mathsf{height}(\boldsymbol{t}) \leq 2|Q|$.*

Proof. For proving emptiness, we assume that L is non-empty and all trees $\boldsymbol{t}$ in L have a height such that $\mathsf{height}(\boldsymbol{t}) > |Q|$. Then, we can take $k = |Q|$ and, following the same arguments as in the proof of the pumping lemma, we can find contexts $\mathbf{c}$ and $\mathbf{c}'$ and a subtree $\boldsymbol{t}'$ for any tree $\boldsymbol{t}$. It follows

that $\mathbf{c}[\mathbf{c}'^n[\boldsymbol{t}']] \in L$ for any $n \geq 0$. For $n = 0$, $\mathbf{c}[\mathbf{c}'^0[\boldsymbol{t}']] = \mathbf{c}[[\boldsymbol{t}']$, which must have a height smaller than k. This leads to a contradiction.

For proving finiteness, we take $k = |Q|$ and assume that for any term $\boldsymbol{t}$ it holds that $|Q| < \mathsf{height}(\boldsymbol{t}) \leq 2|Q|$. Then any term can be pumped as shown the proof of the pumping lemma for any $n \geq 0$, resulting in an infinite number of terms. ■

1.4 Weighted Finite-State Tree Automata

In this section we introduce a generalization of the finite-state tree automaton that we've seen so far, namely the **weighted finite-state tree automaton**. The transitions, as well as the initial and final states have weights from a semiring.

Definition 1.4.1 (Weighted Finite-State Tree Automaton)**.** *A **weighted finite-state tree automaton** (WFSTA) over a semiring $\mathcal{W} = (\mathbb{K}, \oplus, \otimes, \mathbf{0}, \mathbf{1})$ is a 5-tuple $(\Sigma, Q, \lambda, \rho, \delta)$ where:*

- *Q is a finite set of states;*
- *Σ is a ranked alphabet;*
- *$\lambda : Q \to \mathbb{K}$ a weighting function over the set of states Q;*
- *$\rho : Q \to \mathbb{K}$ a weighting function over the set of states Q;*
- *$\delta \subseteq Q^n \times (\Sigma \cup \{\varepsilon\}) \times \mathbb{K} \times Q$ is a finite multi-set of transitions, where $n = \max\{\mathsf{arity}(a) \mid a \in \Sigma\}$.*

Definition 1.4.2 (Regular Weighted Tree Languages)**.** *A weighted tree language is **regular** if and only if it accepted by a weighted finite-state tree automaton.*

We say that a tree $\boldsymbol{t} \in \mathcal{T}(\Sigma)$ is **accepted** or **recognized** by a WFSTA $\mathcal{A}$ if it is accepted by the unweighted version of the automaton and $\boldsymbol{t}$'s weight is non-$\mathbf{0}$ under $\mathcal{A}$.

Definition 1.4.3 (Hyperpath weight)**.** *The weight of a hyperpath $\boldsymbol{\pi}$, denoted by $w(\boldsymbol{\pi})$, is defined inductively as follows:*

- *$w(\boldsymbol{\pi}) = 1$ if $\boldsymbol{\pi} = q, q \in Q$;*
- *$w(\boldsymbol{\pi}) = \bigotimes_{i=1}^{k} w(\boldsymbol{\pi}_i) \otimes w$ if $\boldsymbol{\pi} = \langle \boldsymbol{\pi}_1, \cdots, \boldsymbol{\pi}_k \rangle \xrightarrow{a} q$ and $\langle n(\boldsymbol{\pi}_1), \cdots, n(\boldsymbol{\pi}_k) \rangle \xrightarrow{a/w} q \in \delta$.*

Definition 1.4.4 (Path sum)**.** *Let $\Pi(\mathcal{A})$ be the set of paths in the WFSTA $\mathcal{A}$ over the semiring $\mathcal{W} = (\mathbb{K}, \oplus, \otimes, \mathbf{0}, \mathbf{1})$. The **path sum** of $\mathcal{A}$ is the sum of all hyperpaths in $\mathcal{A}$, defined formally with*

$$Z(\mathcal{A}) = \bigoplus_{\boldsymbol{\pi} \in \Pi(\mathcal{A})} \bigotimes_{q \in p(\boldsymbol{\pi})} \lambda(q) \otimes w(\boldsymbol{\pi}) \otimes \rho(n(\boldsymbol{\pi})). \tag{1.3}$$

Definition 1.4.5 (Tree sum)**.** *The **tree sum** or tree weight of the tree $\boldsymbol{t} \in \mathcal{T}(\Sigma)$ under automaton $\mathcal{A}$ is defined as*

$$\mathcal{A}(\boldsymbol{t}) \stackrel{\text{def}}{=} \bigoplus_{q \in F} \boldsymbol{\alpha}_{\mathcal{A}}(\boldsymbol{t}, q) \otimes \rho(q) \tag{1.4}$$

where $\boldsymbol{\alpha}_{\mathcal{A}} : \mathcal{T}(\Sigma) \times Q \to \mathbb{K}$ is a function that can be computed inductively by

- *$\boldsymbol{\alpha}_{\mathcal{A}}(a, q) = \bigoplus_{p \xrightarrow{a/w} q, \lambda(p) \geq 0} \lambda(p) \otimes w$ for any $a \in \Sigma^0$;*
- *$\boldsymbol{\alpha}_{\mathcal{A}}(\boldsymbol{t}, q) = \bigoplus_{(q_1, q_2, \ldots, q_k) \xrightarrow{a/w} q} \bigotimes_{i=1}^{k} \boldsymbol{\alpha}_{\mathcal{A}}(\boldsymbol{t}_i, q_i) \otimes w$ for any tree such that $\boldsymbol{t} = a\langle \boldsymbol{t}_1, \boldsymbol{t}_2, \ldots, \boldsymbol{t}_k \rangle$.*

The function $\boldsymbol{\alpha}$ is analogous to the forward weight that we've seen for string automata.

1.5 Top-Down WFSTAs

So far we defined bottom-up finite-state tree automata, both in the weighted and the unweighted case, and we've seen a variety of algorithms for transforming them. In this section we turn to their top-down counterparts and make a comparison between the two. We will prove that they recognize the same class of tree languages in the non-deterministic case but are less powerful in the deterministic case.

Definition 1.5.1. *A **top-down finite-state tree automaton** is a 5-tuple $(\Sigma, Q, I, F, \delta)$ where:*

- *Q is a finite set of states;*
- *Σ is a ranked alphabet;*
- *$I \subseteq Q$ is a set of initial states;*
- *$F \subseteq Q$ is a set of final or accepting states;*
- *$\delta \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times Q^n$ is a finite multi-set of transitions, where $n = \max\{\mathsf{arity}(a) \mid a \in \Sigma\}$;*

A top-down FSTA starts its computation at its root in an initial state, then simultaneously moves downwards in all its subtrees following transitions from δ until all its leaves are labeled with states from F. Following the same notation we used for bottom-up FSTAs, we will use hyperedges of the form $q \xrightarrow{a} \langle q_1, q_2, \dots q_k \rangle$. Intuitively, when found in a state q, the top-down FSTA will process the symbol a then move to states $q_1, q_2, \dots q_k$ and process in parallel its immediate children. Similar to the "initial rules" we had in the bottom-up case, here we have transitions of the form $q \xrightarrow{a} q_F$ for each constant symbol $a \in \Sigma^0$ and some state $q_F \in F$.

Definition 1.5.2. *A **weighted top-down finite-state tree automaton** over a semiring $\mathcal{W} = (\mathbb{K}, \oplus, \otimes, \boldsymbol{0}, \boldsymbol{1})$ is a 5-tuple $(\Sigma, Q, \lambda, \rho, \delta)$ where:*

- *Q is a finite set of states;*
- *Σ is a ranked alphabet;*
- *$\lambda : Q \to \mathbb{K}$ a weighting function over the set of states Q;*
- *$\rho : Q \to \mathbb{K}$ a weighting function over the set of states Q;*
- *$\delta \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times \mathbb{K} \times Q^n$ is a finite multi-set of transitions, where $n = \max\{\mathsf{arity}(a) \mid a \in \Sigma\}$.*

Definition 1.5.3 (Tree sum)**.** *The **tree sum** or tree weight of the tree $\boldsymbol{t} \in \mathcal{T}(\Sigma)$ under the top-down FSTA $\mathcal{A}$ is defined as*

$$\mathcal{A}(\boldsymbol{t}) \stackrel{\text{def}}{=} \bigoplus_{\lambda(q) \geq 0} \lambda(q) \otimes \boldsymbol{\beta}_{\mathcal{A}}(\boldsymbol{t}, q) \tag{1.5}$$

where $\boldsymbol{\beta}_{\mathcal{A}} : \mathcal{T}(\Sigma) \times Q \to \mathbb{K}$ is a function that can be computed inductively by

- $\boldsymbol{\beta}_{\mathcal{A}}(a, q) = \bigoplus_{q \xrightarrow{a/w} \langle p \rangle, \rho(p) \geq 0} w \otimes \rho(p)$ *for any $a \in \Sigma^0$;*
- $\boldsymbol{\beta}_{\mathcal{A}}(\boldsymbol{t}, q) = \bigoplus_{q \xrightarrow{a/w} \langle q_1, q_2, \dots, q_k \rangle} w \otimes \bigotimes_{i=1}^{k} \boldsymbol{\beta}_{\mathcal{A}}(\boldsymbol{t}_i, q_i)$ *for any tree such that $\boldsymbol{t} = a\langle \boldsymbol{t}_1, \boldsymbol{t}_2, \dots, \boldsymbol{t}_k \rangle$.*

Definition 1.5.4 (Path sum)**.** *Let* $\Pi(\mathcal{A})$ *be the set of paths in the WFSTA* $\mathcal{A}$ *over the semiring* $\mathcal{W} = (\mathbb{K}, \oplus, \otimes, \mathbf{0}, \mathbf{1})$*. The* ***pathsum*** *of* $\mathcal{A}$ *is the sum of all hyperpaths in* $\mathcal{A}$*, defined formally with*

$$Z(\mathcal{A}) = \bigoplus_{\boldsymbol{\pi} \in \Pi(\mathcal{A})} \lambda(p(\boldsymbol{\pi})) \otimes w(\boldsymbol{\pi}) \otimes \bigotimes_{q \in n(\boldsymbol{\pi})} \rho(q). \tag{1.6}$$

Theorem 1.5.1 (Equivalence of non-deterministic top-down and bottom-up FSTAs)**.** *The class of languages accepted by non-deterministic top-down finite-state tree automata is the class of* ***regular*** *tree languages.*

In other words, this means that non-deterministic top-down and bottom-up FSTAs have equivalent expressive power, i.e. they accept the same tree languages. In fact, we could give a constructive proof that shows that for each non-deterministic top-down weighted FSTA, there exists a bottom-up version that accepts the same tree language.

Definition 1.5.5 (Deterministic top-down FSTA)**.** *A top-down finite-state tree automaton* $\mathcal{A} = (\Sigma, Q, I, F, \delta)$ *is called* ***deterministic*** *or* ***sequential*** *if for every* $(q, a) \in Q \times \Sigma$*,* $\mathsf{arity}(a) = p$*, there exists at most one tuple of states* $\langle q_1, q_2, \ldots q_p \rangle \in Q^p$ *such that* $(q, a, \langle q_1, q_2, \ldots q_p \rangle) \in \delta$ *and there is only one initial state i.e.,* $|I| = 1$*.*

Nondeterministic bottom-up and top-down finite tree automata have equivalent expressive power. As we've seen, they can be easily converted into each other. However, deterministic top-down finite tree automata are strictly less powerful than their bottom-up counterparts.

Theorem 1.5.2. *Deterministic top-down FSTAs are strictly less powerful than non-deterministic top-down FSTAs, i.e. there exists a regular tree language that is not accepted by any deterministic top-down FSTA.*

1.6 WFSTAs and WCFGs

So far in this chapter we've looked at similarities between finite-state *tree* automata and finite-state automata. The majority of the algorithms presented for regular tree languages are straightforward extensions of the algorithms presented in the context of regular string languages. In this section we will look at regular tree languages from a different angle by presenting how they are related to context-free *string*[5] languages.

It turns out that the strings yielded by the trees in a regular tree language form a context-free language. This shouldn't come as a surprise as the rules indeed look very similar. The transitions of a top-down FSTA are of the form $q \xrightarrow{A} \langle q_1, q_2, \cdots, q_k \rangle$ while the productions rules of a context-free grammar look like $A \to \alpha_1 \alpha_2 \cdots \alpha_k$. Before presenting their connection formally, we must emphasize one key difference. The ranked alphabet of the FSTA contains symbols of *fixed* arity, which means that in the top-down case, the right-hand sides of the transitions have a fixed number of arguments for a given symbol. The production rules of context-free grammars do not impose such restrictions, allowing right-hand sides of variable length for the same non-terminal.

Before diving into the theorems, let us consider the following definition of tree automaton constructed from a context-free grammar.

Definition 1.6.1. *Let $\mathfrak{G} = (\mathcal{N}, \Sigma, \mathrm{S}, \mathcal{P})$ be a context-free string grammar. The top-down finite-state tree automaton $\mathcal{A}_{\mathfrak{G}} = (\Sigma_{\mathfrak{G}}, Q_{\mathfrak{G}}, I_{\mathfrak{G}}, F_{\mathfrak{G}}, \delta_{\mathfrak{G}})$ has ranked alphabet $\Sigma = \{A_k \mid A \to \alpha_1 \cdots \alpha_k \in \mathcal{P}\} \cup \Sigma$, states $Q = \{q_A \mid A \in \mathcal{N} \cup \Sigma\} \cup \{q_F\}$, initial state $I = \{q_S\}$, final state $F = \{q_F\}$ and transitions $\delta = \left\{ q_A \xrightarrow{A_k} \langle q_{\alpha_1}, \cdots, q_{\alpha_k} \rangle \mid A \to \alpha_1 \cdots \alpha_k \in \mathcal{P} \right\} \cup \left\{ q_a \xrightarrow{a} \langle q_F \rangle \mid a \in \Sigma \right\}$.*

Example 1.6.1. *Let us look in more detail at this definition through an example. Consider the simple grammar $\mathfrak{G} = (\mathcal{N}, \Sigma, \mathrm{S}, \mathcal{P})$ taken from Comon et al. (2008) that generates arithmetic operations. The grammar has nonterminals $\mathcal{N} = \{S, Op\}$, terminals $\Sigma = \{x, +, *\}$, start symbol $\mathrm{S} = S$ and rules as shown in Fig. 1.8.*

For each terminal and nonterminal symbol in $\Sigma \cup \mathcal{N}$ we introduce a corresponding state and an additional final state. In our example we obtain the set of states $Q_{\mathfrak{G}} = \{q_S, q_{Op}, q_*, q_+, q_x, q_F\}$. We mark the initial state $I_{\mathfrak{G}} = \{q_S\}$ and final state $F_{\mathfrak{G}} = \{q_F\}$. The ranked alphabet is made of the terminals from Σ (constant symbols) and some new special symbols. We introduce a symbol A_k for each right-hand side length of a rule $A \to \alpha \in \mathcal{P}$, $|\alpha| = k$. This is done in order to ensure that the ranked alphabet contains symbols of fixed arity. We thus obtain $\Sigma_{\mathfrak{G}} = \{S_3, S_1, Op_1, x, *, +\}$. Finally, we add transitions corresponding to the rules in $\mathcal{P}$, labeled by these new symbols and transitions leading to final states for the constant symbols. The transitions $\delta_{\mathfrak{G}}$ are shown in Fig. 1.8.

Theorem 1.6.1. *Let $\mathfrak{G} = (\mathcal{N}, \Sigma, \mathrm{S}, \mathcal{P})$ be a context-free grammar that generates the context-free language $L(\mathfrak{G}) = L$. The set of derivation trees of L is a regular tree grammar.*

Notice that the theorem does not imply that there is a one-to-one mapping between regular tree languages and context-free languages. It is often the case that multiple tree languages yield the same context-free language.

We've seen in the previous chapters that the yield of a tree $\boldsymbol{t}$, denoted by $\mathrm{s}(\boldsymbol{t})$, is the concatenation of $\boldsymbol{t}$'s leaves, from left to right. The yield of a tree can also be defined recursively in terms of its subtrees. The following definition formalizes this.

Definition 1.6.2 (Yield)**.** *The **yield** of a tree $\boldsymbol{t}$, denoted by $\mathrm{s}(\boldsymbol{t})$ is defined inductively as follows:*

[5] These are the grammars presented earlier in the course. We emphasize that they are for strings as there exist context-free *tree* grammars as well. These, however, are out of the scope of this course.

$$S \to S\ Op\ S$$
$$S \to x$$
$$Op \to +$$
$$Op \to *$$

(a) Production rules $\mathcal{P}$ of the context-free grammar $\mathfrak{G}$.

$$q_S \xrightarrow{S_3} \langle q_S, q_{Op}, q_S \rangle$$
$$q_S \xrightarrow{S_1} \langle q_x \rangle$$
$$q_{Op} \xrightarrow{Op_1} \langle q_x \rangle$$
$$q_{Op} \xrightarrow{Op_1} \langle q_* \rangle$$
$$q_{Op} \xrightarrow{Op_1} \langle q_+ \rangle$$
$$q_x \xrightarrow{x} \langle q_F \rangle$$
$$q_* \xrightarrow{*} \langle q_F \rangle$$
$$q_+ \xrightarrow{+} \langle q_F \rangle$$

(b) Transitions $\delta_{\mathfrak{G}}$ of the automaton $\mathcal{A}_{\mathfrak{G}}$ constructed from $\mathfrak{G}$.

Figure 1.8: Example of a production rules of a simple context-free grammar $\mathfrak{G}$ together with the transitions of $\mathcal{A}_{\mathfrak{G}}$ constructed as detailed in Def. 1.6.1.

- *If* $\boldsymbol{t} = a \in \Sigma$, $\mathrm{s}(\boldsymbol{t}) = a$;
- *If* $\boldsymbol{t} = a(\boldsymbol{t}_1, \cdots, \boldsymbol{t}_k)$, $\mathrm{s}(\boldsymbol{t}) = \mathrm{s}(\boldsymbol{t}_1) \circ \cdots \circ \mathrm{s}(\boldsymbol{t}_k)$.

Theorem 1.6.2. *Let L be a regular tree language. Then the set of strings $\{\mathrm{s}(\boldsymbol{t}) \mid \boldsymbol{t} \in L\}$ is a context-free string language.*

Theorem 1.6.3. *The class of derivation trees of context-free languages is a strict subset of the class of regular tree languages.*

In other words, there exist regular tree languages that are not collections of derivation trees of any context-free grammar. We can prove this by giving an example of such a tree language, as shown in Example 1.6.2.

Example 1.6.2. *Consider the top-down FSTA $\mathcal{A}$ with ranked alphabet $\Sigma = \{b/0, A/2\}$ set of states, $Q = \{q_0, q_1, q_2, q_3\}$, initial state $I = \{q_0\}$, final state $F = \{q_3\}$ and the following set of transitions $\delta = \left\{q_2 \xrightarrow{b} \langle q_3 \rangle, q_1 \xrightarrow{A} \langle q_2, q_2 \rangle, q_0 \xrightarrow{A} \langle q_1, q_2 \rangle, q_0 \xrightarrow{A} \langle q_2, q_1 \rangle, q_0 \xrightarrow{A} \langle q_1, q_1 \rangle\right\}$. It is easy to see that the language L accepted by this automaton is the set of trees $\{A(A(b,b),b), A(b,A(b,b)), A(A(b,b),A(b,b))\}$, shown in Fig. 1.9.*

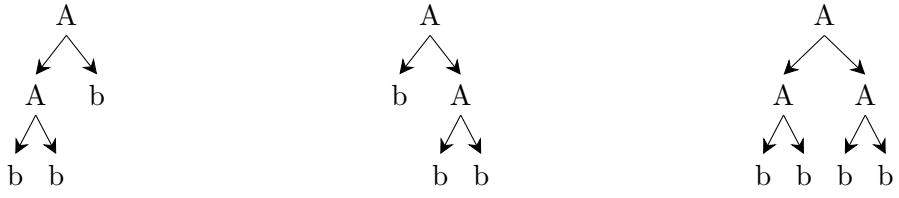


Figure 1.9: The set of trees in the language L accepted by automaton $\mathcal{A}$.

Now let's assume that there is some grammar $\mathfrak{G}$ that can generate this set of parse trees. Then A must be the start symbol and there must be some rule $A \to AA$ so that the third tree from Fig. 1.9 can

be generated. But then the same rule can be re-applied so that we obtain parse trees with arbitrary height, which are not in L.

Lemma 1.6.1. *Derivation trees $\boldsymbol{t} \in \mathcal{D}_{\mathfrak{G}}$ in WCFGs are equivalent to paths $\boldsymbol{\pi} \in \Pi(I, F)$ in WFSTAs.*

Proof. We prove that for any $\boldsymbol{t} \in \mathcal{D}_{\mathfrak{G}}$, $w(\boldsymbol{t}) = w(\boldsymbol{\pi})$ for some path $\boldsymbol{\pi} \in \Pi(I, F)$ in a WFSTA. We define a weighted automaton $\mathcal{A}_{\mathfrak{G}}$ over the semiring $\mathcal{W} = (\mathbb{K}, \oplus, \otimes, \mathbf{0}, \mathbf{1})$ with the same topology as defined in Def. 1.6.1. Additionally, we set the initial weight $\lambda(q_S) = \mathbf{1}$ and final weight $\rho(q_F) = \mathbf{1}$. Finally, we define the weights of the transitions as follows:

$$\left\{ q_A \xrightarrow{A_k/w} \langle q_{\alpha_1}, \cdots, q_{\alpha_k} \rangle \mid A \to \alpha_1 \cdots \alpha_k \in \mathcal{P}, w = w(A \to \alpha_1 \cdots \alpha_k) \right\} \subset \delta$$

$$\left\{ q_a \xrightarrow{a/\mathbf{1}} \langle q_F \rangle \mid a \in \Sigma \right\} \subset \delta.$$

It is easy to show that by setting the weights of $\mathcal{A}_{\mathfrak{G}}$ in this way, the weights of the paths are equivalent to the weights of derivations in $\mathcal{D}_{\mathfrak{G}}$. We leave this as an exercise. ■

Theorem 1.6.4. *Treesums in WCFGs are equivalent to WFSTAs pathsums.*

Proof. Let $\mathfrak{G}$ be a WCFG and $\mathcal{A}_{\mathfrak{G}}$ a WFSTA as defined in Lemma 1.6.1. We have just proven that for any $\boldsymbol{t} \in \mathcal{D}_{\mathfrak{G}}$, $w(\boldsymbol{t}) = w(\boldsymbol{\pi})$ for some path $\boldsymbol{\pi} \in \Pi(I, F)$ in $\mathcal{A}_{\mathfrak{G}}$. Since the initial and final weights are equal to $\mathbf{1}$, it also holds that $w(d) = \lambda(p(\boldsymbol{\pi})) \otimes w(\boldsymbol{\pi}) \otimes \bigotimes_{q \in n(\boldsymbol{\pi})} \rho(q)$. It is easy to see that there are no additional paths from I to F that are not equivalent to some derivation tree of $\mathfrak{G}$. Then it holds that

$$Z_{\mathfrak{G}} = \bigoplus_{\boldsymbol{t} \in \mathcal{D}_{\mathfrak{G}}(S)} w(\boldsymbol{t}) \tag{1.7}$$

$$= \bigoplus_{\boldsymbol{\pi} \in \Pi(I,F)} \lambda(p(\boldsymbol{\pi})) \otimes w(\boldsymbol{\pi}) \otimes \bigotimes_{q \in n(\boldsymbol{\pi})} \rho(q) \tag{1.8}$$

$$= Z(\mathcal{A}) \tag{1.9}$$

This concludes the proof. ■

1.7 ε-Removal

Similar to finite-state automata, tree automata can also have ε-transitions of the form $p \xrightarrow{\varepsilon/w} q$. Note that we treat the special symbol ε as a unary symbol (arity 1) from our ranked alphabet. Recall that a ε-transitions enable a finite-state automaton to move from one state to another state without consuming a symbol of the string. Similarly, ε-transitions enable tree automata to move to another state without processing any symbol of the term/tree. Just like in the case of finite-state automata, ε-transitions do not change the language accepted by the automaton but they are useful in certain constructions and simplify some proofs.

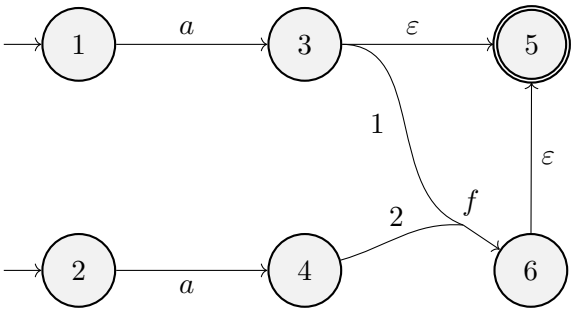


Figure 1.10: Example of FSTA with ε transitions.

Theorem 1.7.1. *For any (weighted) finite-state tree automaton with ε-rules, there exists a (weighted) finite-state tree automaton without ε-rules that recognizes the same (weighted) tree language.*

In other words, we can come up with an algorithm that converts any WFSTA into an equivalent one that is ε-free. By equivalent we mean a WFSTA that recognizes the same weighted tree language. It turns out that we can use the same algorithm that we used for WFSAs to do ϵ-removal on a WFSTA as well.

Recall how we computed the ε-closure of a finite-state automaton: we first constructed a new automaton by keeping only the ε-transitions, then we computed the Kleene closure of its transition matrix using Lehmann's algorithm. Now you may wonder how you can compute the ε-closure of a tree automaton. The key detail to notice here is what the ε-transitions look like: the left-hand side of the transition contains a single state, just like the FSA ε-transitions! This means that when constructing a new automaton from the ε-transitions of a WFSTA we actually get a WFSA. Therefore, we can compute the ε-closure of a finite-state tree automaton $\mathcal{A}$ by following the same two steps that we've seen in the context of finite-state automata:

(i) We construct the finite-state automaton $\mathcal{A}_\varepsilon$ by keeping *only* the ε-transitions in the finite-state tree automaton $\mathcal{A}$;

(ii) We compute the Kleene closure of $\mathcal{A}_\varepsilon$'s transition matrix using Lehmann's algorithm.

Algorithm 1 The ϵ-removal algorithm.

1. **def** ϵ-removal($\mathcal{A}$):
2. **compute** the ε-closure $\kappa(\cdot, \cdot)$ ▷*Requires Lehmann's algorithm; runs in* $\mathcal{O}\left(|Q|^3\right)$
3. **initialize** $\mathcal{A}_{\neg\varepsilon}$ as a copy of $\mathcal{A}$ *without* ε-transitions ▷*Initial weights copied, final weights not*
4. **for** $\langle q_1, q_2, \ldots q_k \rangle \xrightarrow{a/w} q \in \delta$:
5. **for** $q' \in Q$:
6. $w' \leftarrow w \otimes \kappa\left(q, q'\right)$
7. **replace** transition $\langle q_1, q_2, \ldots q_k \rangle \xrightarrow{a/w} q$ with $\langle q_1, q_2, \ldots q_k \rangle \xrightarrow{a/w'} q'$ in $\mathcal{A}_{\neg\varepsilon}$
8. **for** $q' \in Q$:
9. $\rho'\left(q'\right) \leftarrow \kappa(q', q) \otimes \bigoplus_{q \in Q} \rho\left(q\right)$
10. **return** $\mathcal{A}_{\neg\varepsilon}$

1.8 Determinization

We will start with an intuitive explanation of what determinism means for FSTAs. Recall how in the case of finite-state automata, determinism implies that when being in some state q and processing a symbol a, there exists a single state we can end up in. As a result, the automaton can be in a single state at one time. A tree automaton, on the other hand, already moves in parallel on each subtree until it reaches the tree's root, thus it is in one state for each subtree it is currently processing. However, the notion of determinism that we saw for FSAs extends naturally to tree automata. When we've already processed subtrees whose roots are in states $\langle q_1, q_2, \ldots, q_k \rangle$ and we move up the tree to their parent node, labeled with an arbitrary symbol $a \in \Sigma$, there is a single state q the parent can be in. This implies that there is at most one run of a deterministic FSTA on each input tree. If we think of the FSTA as a hypergraph, this is equivalent to having at most one hyperpath from the initial states to the final states for each ground term.

Just like we did for FSAs, we use the powerset construction in the determinization procedure of an FSTA. In this context, it means that each node of the tree can be in a subset of the states of the original automaton. Obviously, the determinized automaton will have the same ranked alphabet as the automaton that is being determinized, i.e. $\Sigma_{\text{DET}} = \Sigma$. Just like for FSAs, the state space of the determinized machine will contain powerstates $\mathcal{Q} \in 2^Q$. The determinized automaton can have a single initial state, thus we collapse all initial states in one powerstate $\mathcal{Q}_I$. Finally, we mark a powerstate $\mathcal{Q} = \{\ldots, q_k, \ldots\}$ as final if it contains a state q_k that is final in the original automaton. Now it only remains to describe how we add transitions between powerstates. We add a transition $\langle \mathcal{Q}_1, \mathcal{Q}_2, \ldots, \mathcal{Q}_k \rangle \xrightarrow{a} \mathcal{Q}$ if and only if there exist states $q \in \mathcal{Q}$, $q_1 \in \mathcal{Q}_1$, $q_2 \in \mathcal{Q}_2$, $\ldots$, $q_k \in \mathcal{Q}_k$ such that there is a transition $\langle q_1, q_2, \ldots, q_k \rangle \xrightarrow{a} q$ in the machine being determinized.

Obviously, listing all these combinations of states and checking whether the condition holds is a very expensive operation, thus we will develop an on-the-fly version of the determinization algorithm. So far, the construction that we described has been a straightforward extension of the determinization procedure for finite-state automata. There are, however, a few important adjustments that we need to make to Algorithm **??** so that is works for tree automata.

(i) Recall how the algorithm for strings uses a stack to keep track of unexplored accessible power states and expands them sequentially, adding at each step the set of states that can be reached from the current state. A FSTA, however, can be in multiple states at one same time, processing different subtrees in parallel. For this reason, we modify our stack so that it tracks

the unexplored *vectors of power states* in which the FSTA can simultaneously be.

(ii) The second change that we need to make concerns the creation of new vectors of power states to add to the stack. For each newly-added power state $\mathcal{Q}$, we create accessible vectors of power states by listing all ordered combinations of already-existing states from Q_{DET} and $\mathcal{Q}$. We create vectors of power states of all possible lengths up the maximum arity of the symbols from Σ as we cannot have transitions with left-hand sides beyond this length. This is done by the combinations function in Line 14.

(iii) We change the arities of the constant symbols from 0 to 1 such that their arities match the length of the left-hand sides of the transitions they are involved in.

The whole procedure is outlined in Algorithm 2, following May and Knight (2006). The meaning of $\mathcal{E}_{\mathcal{A}}(\langle q_1, q_2, \ldots, q_k\rangle)$ is similar to the one used for finite-state automata, namely the set of outgoing hyperedges leaving from states $\langle q_1, q_2, \ldots, q_k\rangle$, i.e. $\mathcal{E}_{\mathcal{A}}(\langle q_1, q_2, \ldots, q_k\rangle) = \{(\langle q_1, q_2, \ldots, q_k\rangle, a, q) \in \delta\}$. Additionally, we overload $\mathcal{E}$ for sets of states $\mathcal{Q} \subseteq Q$ as follows:

$$\mathcal{E}_{\mathcal{A}}(\langle \mathcal{Q}_1, \mathcal{Q}_2, \ldots, \mathcal{Q}_k\rangle) = \{(\langle q_1, q_2, \ldots, q_k\rangle, a, q) \in \delta | q_1 \in \mathcal{Q}_1, q_2 \in \mathcal{Q}_2, \ldots, q_k \in \mathcal{Q}_k\}. \tag{1.10}$$

Algorithm 2 The FSTA Determinize algorithm.

1. **def** Determinize($\mathcal{A}$):
2. $\quad\mathcal{A}_{\text{DET}} \leftarrow (\Sigma_{\text{DET}}, Q_{\text{DET}}, I_{\text{DET}}, F_{\text{DET}}, \delta_{\text{DET}})$ ▷*Initialize new FSTA.*
3. $\quad\mathcal{Q}_I \leftarrow \{q_I | q_I \in I\}$
4. $\quad$stack $\leftarrow \{\langle \mathcal{Q}_I\rangle\}$ ▷*Add vector containing the initial state to* stack
5. $\quad$visited $\leftarrow \emptyset$
6. $\quad Q_{\text{DET}} \leftarrow \{\mathcal{Q}_I\}$ ▷*Add the initial state to* Q_{DET}
7. $\quad$**while** |stack| > 0 :
8. $\quad\quad$**pop** vector of power states $\langle \mathcal{Q}_1, \mathcal{Q}_2, \ldots, \mathcal{Q}_k\rangle$ from stack
9. $\quad\quad$visited $\leftarrow$ visited $\cup \{\langle \mathcal{Q}_1, \mathcal{Q}_2, \ldots, \mathcal{Q}_k\rangle\}$
10. $\quad\quad$**for** $a \in$ symbols $(\mathcal{E}_{\mathcal{A}}(\langle \mathcal{Q}_1, \mathcal{Q}_2, \ldots, \mathcal{Q}_k\rangle))$:
11. $\quad\quad\quad\mathcal{Q}' \leftarrow \left\{q : \bullet \xrightarrow{a} q \in \mathcal{E}_{\mathcal{A}}(\langle \mathcal{Q}_1, \mathcal{Q}_2, \ldots, \mathcal{Q}_k\rangle)\right\}$ ▷*Construct a new power state.*
12. $\quad\quad\quad\delta_{\text{DET}} \leftarrow \delta_{\text{DET}} \cup \left\{\langle \mathcal{Q}_1, \mathcal{Q}_2, \ldots, \mathcal{Q}_k\rangle \xrightarrow{a} \mathcal{Q}'\right\}$
13. $\quad\quad\quad$**if** $\mathcal{Q}' \notin$ visited :
14. $\quad\quad\quad\quad$**if** $\mathcal{Q}' \cap F \neq \emptyset$:
15. $\quad\quad\quad\quad\quad F_{\text{DET}} \leftarrow F_{\text{DET}} \cup \{\mathcal{Q}'\}$
16. $\quad\quad\quad\quad$▷*combinations* $(\mathcal{Q}, N, k)$ *returns all possible tuples of length* $1 \ldots k$ *containing members of* $\mathcal{Q}$ *and at least one member of* N; *rank* (q) *returns the largest hyperedge size that can leave state* q
17. $\quad\quad\quad\quad$**for** $\langle \mathcal{Q}_1, \mathcal{Q}_2, \ldots, \mathcal{Q}_p\rangle \in$ combinations $\left(Q_{\text{DET}}, \{\mathcal{Q}'\}, \max_{q \in \{p | p \in Q_{\text{DET}} \cup \{\mathcal{Q}'\}\}} \text{rank}(q)\right)$:
18. $\quad\quad\quad\quad\quad$**if** $\langle \mathcal{Q}_1, \mathcal{Q}_2, \ldots, \mathcal{Q}_p\rangle \notin$ visited :
19. $\quad\quad\quad\quad\quad\quad$**push** $\langle \mathcal{Q}_1, \mathcal{Q}_2, \ldots, \mathcal{Q}_p\rangle$ onto stack
20. $\quad\quad\quad\quad Q_{\text{DET}} \leftarrow Q_{\text{DET}} \cup \{\mathcal{Q}'\}$
21. $\quad$**return** $\mathcal{A}_{\text{DET}}$

Lemma 1.8.1. *Let* $\mathcal{A}_{\text{DET}} =$ Determinize$(\mathcal{A})$ *be the output of* Determinize *on the unweighted FSTA* $\mathcal{A}$. *Then,* $\mathcal{A}_{\text{DET}}$ *is deterministic.*

Proof. First note that the determinized automaton has a single initial state which contains all the elementary initial states in the original automaton (Line 3) and no ε-transitions are added during the determinization algorithm. Now it suffices to prove that for each vector of powerstates and symbol, there is a single outgoing transition in the determinized automaton. Each vector of power states can be added to the stack at most once (after it is popped from the stack, it is added to the set of visited vectors and cannot be added again to the stack because of the check in Line 18). For each vector of source power states, a single transition is added per symbol (Line 12). ■

An important consequence of an automaton being deterministic is that for each term $\boldsymbol{t}$ there is at most one hyperpath yielding $\boldsymbol{t}$. This is perhaps not entirely obvious from the definition of a deterministic FSTA, thus we prove it in the following lemma.

Lemma 1.8.2. *For any term $\boldsymbol{t}$ that is accepted by $\mathcal{A}$, there is a unique hyperpath yielding $\boldsymbol{t}$ from I to F in the determinized automaton $\mathcal{A}_{\text{DET}}$.*

In order to prove the correctness of the algorithm, we need to show that the output automaton is equivalent to the original one. We will start by proving that the power state we end up in by following the unique hyperpath yielding $\boldsymbol{t}$ in $\mathcal{A}_{\text{DET}}$ contains the elementary states we end up in by following all the hyperpaths yielding $\boldsymbol{t}$ in the original automaton.

Before formalizing this statement in a lemma, let us introduce some new notation. We use $\delta(\mathcal{Q}, \boldsymbol{t})$ to denote the set of states that are reachable by following any hyperpath starting from the set of states $\mathcal{Q} \subseteq I$ that yields $\boldsymbol{t}$.

Lemma 1.8.3. *Let $\mathcal{A}_{\text{DET}} = (\Sigma_{\text{DET}}, Q_{\text{DET}}, I_{\text{DET}}, F_{\text{DET}}, \delta_{\text{DET}})$ be the output of Determinize on the automaton $\mathcal{A}$. Assume moreover that $\mathcal{Q}$ is the power state reached when following the unique hyperpath yielding $\boldsymbol{t}$. Then it holds that*

$$\mathcal{Q} = \delta(I, \boldsymbol{t}). \tag{1.11}$$

Proof. We will prove the lemma by induction on the height of the trees.

Base Case. Let $\boldsymbol{t}$ have a height of 1, i.e. $\boldsymbol{t} = a, a \in \Sigma^0$. $\mathcal{Q}_I$ is the initial state of the determinized automaton and $\mathcal{Q}' \leftarrow \left\{q : \bullet \xrightarrow{a} q \in \mathcal{E}_{\mathcal{A}}(\langle \mathcal{Q}_I \rangle)\right\}$ is the powerstate reached by following a transition labeled with a.

$$\begin{aligned}
\mathcal{Q}' &\leftarrow \left\{q : \bullet \xrightarrow{a} q \in \mathcal{E}_{\mathcal{A}}(\langle \mathcal{Q}_I \rangle)\right\} && (1.12)\\
&= \left\{q : q_I \xrightarrow{a} q, q_I \in I\right\} && (1.13)\\
&= \delta(I, \boldsymbol{t}) && (1.14)
\end{aligned}$$

Inductive Step. Let $\boldsymbol{t} = a(\boldsymbol{t}_1, \cdots, \boldsymbol{t}_k)$, where $\max_{i \in [1,k]} \mathsf{height}(\boldsymbol{t}_i) = n$ and let $\mathcal{Q}_i$ be the powerstate reached by following the hyperpath yielding $\boldsymbol{t}_i$. By the induction hypothesis, it holds that $\mathcal{Q}_i = \delta(I, \boldsymbol{t}_i), \forall i \in [1, k]$. Moreover, since $\mathcal{A}_{\text{DET}}$ is deterministic, there is a single state $\mathcal{Q}'$ such that $\langle \mathcal{Q}_1, \cdots, \mathcal{Q}_k \rangle \xrightarrow{a} \mathcal{Q}'$. Any hyperpath yielding $\boldsymbol{t}$ in the original automaton must pass through states $q_1, \cdots, q_k$ and have a final transition of the form $\langle q_1, \cdots, q_k \rangle \xrightarrow{a} q'$. Formally,

$$\begin{aligned}
\mathcal{Q}' &\leftarrow \left\{q' : \bullet \xrightarrow{a} q' \in \mathcal{E}_{\mathcal{A}}(\langle \mathcal{Q}_1, \cdots, \mathcal{Q}_k \rangle)\right\} && (1.15)\\
&= \left\{q' : \langle q_1, \cdots, q_k \rangle \xrightarrow{a} q', q_1 \in \mathcal{Q}_1, \cdots, q_k \in \mathcal{Q}_k\right\} && (1.16)\\
&= \left\{q' : \langle q_1, \cdots, q_k \rangle \xrightarrow{a} q', q_1 \in \delta(I, \boldsymbol{t}_1), \cdots, q_k \in \delta(I, \boldsymbol{t}_k)\right\} && (1.17)\\
&= \delta(I, \boldsymbol{t}). && (1.18)
\end{aligned}$$

This concludes the proof. ■

Theorem 1.8.1. *Let $\mathcal{A} = (\Sigma, Q, I, F, \delta)$ be an ε-free FSA. The output of $\mathcal{A}_{\text{DET}} = \textit{Determinize}(\mathcal{A}) = (\Sigma_{\text{DET}}, Q_{\text{DET}}, I_{\text{DET}}, F_{\text{DET}}, \delta_{\text{DET}})$ accepts the same language as $\mathcal{A}$.*

Proof. This proof follows closely the proof for string automata. ■

Now that we presented the determinization algorithm for unweighted FSTAs, we can extend it for their weighted counterparts. In order to do this, we need to use the concept of residual weights like we did for WFSAs, which we will modify to account for the existence of hyperedges. We assume that the semirings that we use are *divisible* and *commutative*. We define the residual weights formally as follows:

Definition 1.8.1 (Residual weight)**.** *Let $\mathcal{A}_{\text{DET}}$ be deterministic WFSTA over the ranked alphabet Σ and a semiring $\mathcal{W}$. Let $\mathcal{Q}'_1, \mathcal{Q}'_2, \ldots, \mathcal{Q}'_k$ be $\mathcal{A}_{\text{DET}}$'s power states reached after processing subterms $\boldsymbol{t}_1, \boldsymbol{t}_2, \ldots, \boldsymbol{t}_k$ and let $\langle \mathcal{Q}'_1, \mathcal{Q}'_2, \ldots, \mathcal{Q}'_k \rangle \xrightarrow{b/w} \mathcal{Q}$ be a transition in $\mathcal{A}_{\text{DET}}$. Thus, $\mathcal{Q}$ is the unique power state reached after processing the term $b\langle \boldsymbol{t}_1, \boldsymbol{t}_2, \ldots, \boldsymbol{t}_k \rangle$. The **residual weight** (or just **residual**) of the state q in the power state $\mathcal{Q}$ is recursively defined in terms of $\mathcal{Q}'_1$'s, $\mathcal{Q}'_2$'s, ... and $\mathcal{Q}'_k$'s residuals as*

$$r_{\mathcal{Q}}(q) \stackrel{\text{def}}{=} \bigoplus_{q'_1 \in \mathcal{Q}'_1, q'_2 \in \mathcal{Q}'_2, \ldots, q'_k \in \mathcal{Q}'_k} \bigoplus_{\langle q'_1, q'_2, \ldots, q'_k \rangle \xrightarrow{b/w} q \in \mathcal{E}(\langle q'_1, q'_2, \ldots, q'_k \rangle)} w'^{-1} \otimes \bigotimes_{i=1}^{k} r_{\mathcal{Q}'_i}(q'_i) \otimes w \tag{1.19}$$

where we define

$$w' \stackrel{\text{def}}{=} \bigoplus_{q'_1 \in \mathcal{Q}'_1, q'_2 \in \mathcal{Q}'_2, \ldots, q'_k \in \mathcal{Q}'_k} \bigoplus_{\langle q'_1, q'_2, \ldots, q'_k \rangle \xrightarrow{b/w} p \in \mathcal{E}(\langle q_1, q_2, \ldots, q_k \rangle)} \bigotimes_{i=1}^{k} r_{\mathcal{Q}'_i}(q'_i) \otimes w. \tag{1.20}$$

There are three differences between the definition of residuals given in Chapter **??** and our modified version. The first difference is the first $\oplus$-sum, which ranges over all possible combinations of elementary states from a vector of power states. Intuitively, we want to choose one elementary state q from each power state $\mathcal{Q}$, resulting in a vector $\langle q'_1, q'_2, \ldots, q'_k \rangle$, then sum the weights of all transitions of the form $\langle q'_1, q'_2, \ldots, q'_k \rangle \xrightarrow{a/w} q$ in the original automaton.

Example 1.8.1. *Consider two power states $\mathcal{Q}_1 = \{(q_1, 1)\}$ and $\mathcal{Q}_2 = \{(q_1, 0.25), (q_2, 0.75)\}$. Then the two valid vectors of elementary states that can be constructed from $\langle \mathcal{Q}_1, \mathcal{Q}_2 \rangle$ are $\langle (q_1, 1), (q_1, 0.25) \rangle$ and $\langle (q_1, 1), (q_2, 0.75) \rangle$.*

It should be obvious why we sum over transitions of this form in the second sum of the equations. The final modification is the $\otimes$-multiplication of the residual weights of the chosen elementary weights. Based on this definition, we can give an interpretation of the residual weights. The residual weight at state q is the additional weight of the tree sum up to state q that is not accounted for by the tree sums up to the states $q'_1, q'_2, \ldots, q'_k$.

Lemma 1.8.4. *Let $\mathcal{A}_{\text{DET}}$ be a determinized automaton and $\langle \mathcal{Q}'_1, \cdots, \mathcal{Q}'_k \rangle \xrightarrow{a} \mathcal{Q}$ a transition in $\mathcal{A}_{\text{DET}}$. Then*

$$\bigoplus_{q \in \mathcal{Q}} r_{\mathcal{Q}}(q) = \boldsymbol{1}. \tag{1.21}$$

Proof.

$$\bigoplus_{q\in\mathcal{Q}} r_{\mathcal{Q}}(q) = \bigoplus_{q\in\mathcal{Q}} \bigoplus_{q'_1\in\mathcal{Q}'_1,q'_2\in\mathcal{Q}'_2,\ldots,q'_k\in\mathcal{Q}'_k} \bigoplus_{\langle q'_1,q'_2,\ldots,q'_k\rangle \xrightarrow{a/w} q\in\mathcal{E}(\langle q'_1,q'_2,\ldots,q'_k\rangle)} w'^{-1}\otimes \bigotimes_{i=1}^{k} r_{\mathcal{Q}'_i}(q'_i)\otimes w \tag{1.22}$$

$$= w'^{-1}\otimes \bigoplus_{q\in\mathcal{Q}} \bigoplus_{q'_1\in\mathcal{Q}'_1,q'_2\in\mathcal{Q}'_2,\ldots,q'_k\in\mathcal{Q}'_k} \bigoplus_{\langle q'_1,q'_2,\ldots,q'_k\rangle \xrightarrow{a/w} q\in\mathcal{E}(\langle q'_1,q'_2,\ldots,q'_k\rangle)} \bigotimes_{i=1}^{k} r_{\mathcal{Q}'_i}(q'_i)\otimes w \tag{1.23}$$

$$= w'^{-1}\otimes \bigoplus_{q'_1\in\mathcal{Q}'_1,q'_2\in\mathcal{Q}'_2,\ldots,q'_k\in\mathcal{Q}'_k} \bigoplus_{\langle q'_1,q'_2,\ldots,q'_k\rangle \xrightarrow{a/w} p\in\mathcal{E}(\langle q'_1,q'_2,\ldots,q'_k\rangle)} \bigotimes_{i=1}^{k} r_{\mathcal{Q}'_i}(q'_i)\otimes w \tag{1.24}$$

$$= w'^{-1}\otimes w' \tag{1.25}$$

$$= \mathbf{1} \tag{1.26}$$

■

Before presenting the weighted determinization algorithm, we make one final change in notation. We define the projection to a vector of states as

$$\mathcal{E}_{\langle Q_1,Q_2,\ldots,Q_k\rangle}(\langle\mathcal{Q}_1,\mathcal{Q}_2,\ldots,\mathcal{Q}_k\rangle) \stackrel{\text{def}}{=} \langle\mathcal{E}_{Q_1}(\mathcal{Q}_1),\mathcal{E}_{Q_1}(\mathcal{Q}_2),\ldots,\mathcal{E}_{Q_k}(\mathcal{Q}_k)\rangle \tag{1.27}$$

where $\mathcal{E}_Q(\mathcal{Q})$ is as defined in Chapter **??**. In words, the projection returns the vector of power states without the residuals associated to each elementary states.

Example 1.8.2. *Alg. 3 shows an example of a non-deterministic FSTA, together with the equivalent determinized version.*

In the unweighted case the algorithm always terminates and returns a correctly determinized algorithm. The WeightedDeterminize algorithm does not always halt, but when it does it always returns a deterministic automaton that preserves the semantics of the original automaton.

Lemma 1.8.5. *Whenever WeightedDeterminize halts on automaton $\mathcal{A}$, the output automaton $\mathcal{A}_{\text{DET}}$ is deterministic.*

Proof. The lemma can be proven using the same arguments as in the unweighted case. ■

Lemma 1.8.6. *Let $\mathcal{A}_{\text{DET}}$ be the output of WeightedDeterminize on the automaton $\mathcal{A}$ over the divisible semiring $\mathcal{W}$. Let $\boldsymbol{t}$ be a tree and $\mathcal{Q}\in Q_{\text{DET}}$ the powerstate reached by following the unique hyperpath yielding $\boldsymbol{t}$. Then for any elementary state $q\in\mathcal{Q}$ it holds that*

$$\boldsymbol{\alpha}_{\mathcal{A}}(\boldsymbol{t},q) = \boldsymbol{\alpha}_{\mathcal{A}_{\text{DET}}}(\boldsymbol{t},\mathcal{Q})\otimes r_{\mathcal{Q}}(q). \tag{1.28}$$

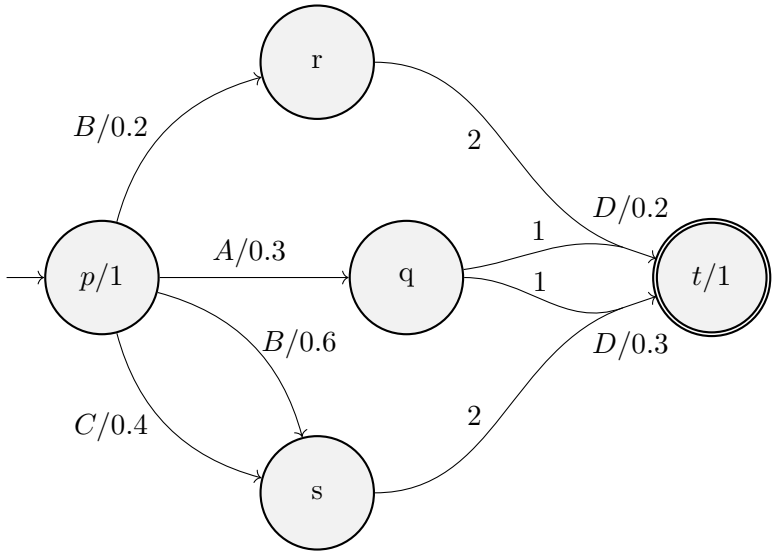
Proof. The proof proceeds by induction over the height of the tree $\boldsymbol{t}$.

Base Case. Assume $\mathsf{height}(\boldsymbol{t}) = 1$, i.e. $\boldsymbol{t} = a, a\in\Sigma^0$. Let $\mathcal{Q}_I$ be the initial state of the

Algorithm 3 The FSTA WeightedDeterminize algorithm.

def WeightedDeterminize($\mathcal{A}$):
  $\mathcal{A}_{\text{DET}} \leftarrow (\Sigma, Q_{\text{DET}}, \lambda_{\text{DET}}, \rho_{\text{DET}}, \delta_{\text{DET}})$ ▷*Initialize new WFSTA over the same semiring as* $\mathcal{A}$.
  $\mathcal{Q}_I \leftarrow \{(q_I, \lambda(q_I)) \,|\, \lambda(q_I) > 0\}$
  $\lambda_{\text{DET}}(\mathcal{Q}_I) \leftarrow \mathbf{1}$
  stack $\leftarrow \{\langle \mathcal{Q}_I \rangle\}$ ▷*Add vector containing the initial state to* stack.
  visited $\leftarrow \emptyset$
  $Q_{\text{DET}} \leftarrow \{\mathcal{Q}_I\}$ ▷*Add the initial state to* Q_{DET}.
  while |stack| > 0 :
    pop vector of power states $\langle \mathcal{Q}_1, \mathcal{Q}_2, \ldots, \mathcal{Q}_k \rangle$ from stack
    visited $\leftarrow$ visited $\cup \{\langle \mathcal{Q}_1, \mathcal{Q}_2, \ldots, \mathcal{Q}_k \rangle\}$
    for $a \in$ symbols $(\mathcal{E}_{\mathcal{A}}(\langle \mathcal{Q}_1, \mathcal{Q}_2, \ldots, \mathcal{Q}_k \rangle))$:
      $w' \leftarrow \bigoplus_{p_1 \in \mathcal{Q}_1, p_2 \in \mathcal{Q}_2, \ldots, p_k \in \mathcal{Q}_k} \bigoplus_{\langle p_1, p_2, \ldots, p_k \rangle \xrightarrow{a/w} q \in \mathcal{E}(\langle p_1, p_2, \ldots, p_k \rangle)} \bigotimes_{i=1}^{k} r_{\mathcal{Q}_i}(p_i) \otimes w$ ▷*Normalizer weight.*
      $\mathcal{Q}' \leftarrow \left\{ \left(q, \bigoplus_{p_1 \in \mathcal{Q}_1, p_2 \in \mathcal{Q}_2, \ldots, p_k \in \mathcal{Q}_k} \bigoplus_{\langle p_1, p_2, \ldots, p_k \rangle \xrightarrow{a/w} q \in \mathcal{E}(\langle p_1, p_2, \ldots, p_k \rangle)} w'^{-1} \otimes \bigotimes_{i=1}^{k} r_{\mathcal{Q}_i}(p_i) \otimes w \right) : \right.$
      $\left. \bullet \xrightarrow{a/\bullet} q \in \mathcal{E}(\mathcal{E}_{Q_1, Q_2, \ldots, Q_k}(\langle \mathcal{Q}_1, \mathcal{Q}_2, \ldots, \mathcal{Q}_k \rangle)) \right\}$ ▷*New power state.*
      $\delta_{\text{DET}} \leftarrow \delta_{\text{DET}} \cup \left\{ \langle \mathcal{Q}_1, \mathcal{Q}_2, \ldots, \mathcal{Q}_k \rangle \xrightarrow{a/w'} \mathcal{Q}' \right\}$
      if $\mathcal{Q}' \notin$ visited :
        $\rho_{\text{DET}}(\mathcal{Q}') \leftarrow \bigotimes_{q \in \mathcal{Q}'} r_{\mathcal{Q}'}(q) \otimes \rho(q)$
        ▷*combinations* $(\mathcal{Q}, N, k)$ *returns all possible tuples of length* $1 \ldots k$ *containing members of* $\mathcal{Q}$ *and at least one member of* N; *rank* (q) *returns the largest hyperedge size that can leave state* q
        for $\langle \mathcal{Q}_1, \mathcal{Q}_2, \ldots, \mathcal{Q}_p \rangle \in$ combinations $\left(Q_{\text{DET}}, \{\mathcal{Q}'\}, \max_{q \in \{p | p \in Q_{\text{DET}} \cup \{\mathcal{Q}'\}\}} \text{rank}(q) \right)$:
          if $\langle \mathcal{Q}_1, \mathcal{Q}_2, \ldots, \mathcal{Q}_p \rangle \notin$ visited :
            push $\langle \mathcal{Q}_1, \mathcal{Q}_2, \ldots, \mathcal{Q}_p \rangle$ onto stack
        $Q_{\text{DET}} \leftarrow Q_{\text{DET}} \cup \{\mathcal{Q}'\}$
  return $\mathcal{A}_{\text{DET}}$

(a) Example of non-deterministic finite-state tree automaton.



(b) Determinized version of the automaton.

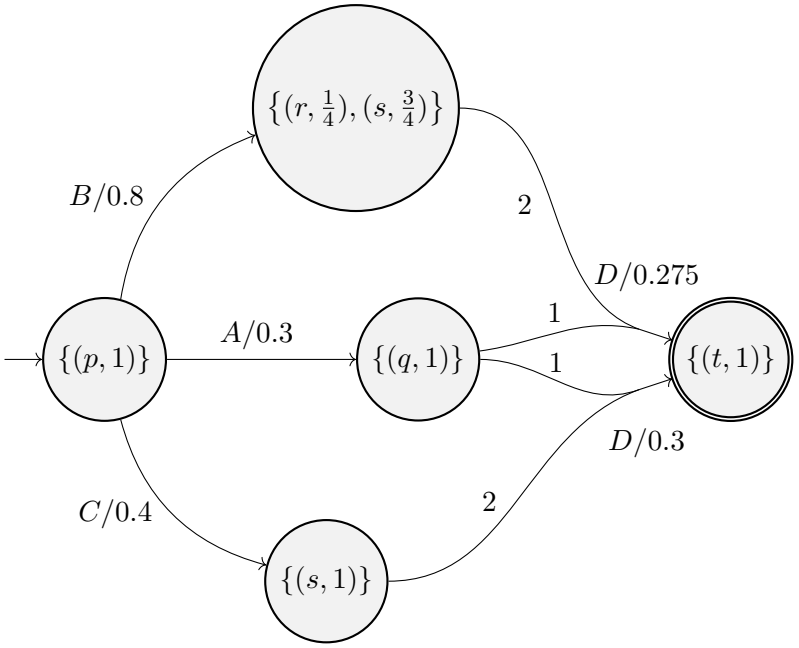


Figure 1.11: Example of non-deterministic WFSTA together with the determinized version. The numbers on the hyperedges indicate the index of the source node in the left-hand side of the transition.

determinized FSTA and $\mathcal{Q}$ the power state reached by following the transition $\mathcal{Q}_I \xrightarrow{a/w'} \mathcal{Q}$.

$$\begin{aligned}
\boldsymbol{\alpha}_{\mathcal{A}}(\boldsymbol{t}, q) &= \boldsymbol{\alpha}_{\mathcal{A}}(b, q) && (1.29)\\
&= \bigoplus_{q_I \in I} \lambda(q_I) \otimes \bigoplus_{q_I \xrightarrow{a/w} q} w && (1.30)\\
&= \bigoplus_{q_I \in I} r_{\mathcal{Q}_I}(q_I) \otimes \bigoplus_{q_I \xrightarrow{a/w} q} w && (1.31)\\
&= w' \otimes w'^{-1} \otimes \bigoplus_{q_I \in I} r_{\mathcal{Q}_I}(q_I) \otimes \bigoplus_{q_I \xrightarrow{a/w} q} w && (1.32)\\
&= w' \otimes \bigoplus_{q_I \in I} \bigoplus_{q_I \xrightarrow{a/w} q} w'^{-1} \otimes r_{\mathcal{Q}_I}(q_I) \otimes w && (1.33)\\
&= w' \otimes r_{\mathcal{Q}}(q) && (1.34)\\
&= \mathbf{1} \otimes w' \otimes r_{\mathcal{Q}}(q) && (1.35)\\
&= \lambda(\mathcal{Q}) \otimes w' \otimes r_{\mathcal{Q}}(q) && (1.36)\\
&= \boldsymbol{\alpha}_{\mathcal{A}_{\text{DET}}}(\boldsymbol{t}, \mathcal{Q}) \otimes r_{\mathcal{Q}}(q) && (1.37)
\end{aligned}$$

Inductive Step. Assume $\boldsymbol{t} = a(\boldsymbol{t}_1, \cdots, \boldsymbol{t}_k)$ and let $\mathcal{Q}_1, \cdots, \mathcal{Q}_k$ be the powerstates reached by following the paths yielding $\boldsymbol{t}_1, \cdots, \boldsymbol{t}_k$ in $\mathcal{A}_{\text{DET}}$. Assume moreover that $\mathcal{Q}'$ is the power state reached by following the transition $\langle \mathcal{Q}_1, \cdots, \mathcal{Q}_k \rangle \xrightarrow{a/w'} \mathcal{Q}'$.

$$\boldsymbol{\alpha}_{\mathcal{A}}\left(\boldsymbol{t}, q'\right) = \bigoplus_{q_1,\cdots,q_k \in Q} \bigotimes_{i=1}^{k} \boldsymbol{\alpha}_{\mathcal{A}}\left(\boldsymbol{t}_i, q_i\right) \otimes \bigoplus_{\langle q_1,\cdots,q_k\rangle \xrightarrow{a/w} q'} w \tag{1.38}$$

$$= \bigoplus_{q_1,\cdots,q_k \in Q} \bigotimes_{i=1}^{k} \boldsymbol{\alpha}_{\mathcal{A}_{\text{DET}}}\left(\boldsymbol{t}_i, q_i\right) \otimes r_{\mathcal{Q}}\left(q_i\right) \otimes \bigoplus_{\langle q_1,\cdots,q_k\rangle \xrightarrow{a/w} q'} w \tag{1.39}$$

$$= \bigotimes_{i=1}^{k} \boldsymbol{\alpha}_{\mathcal{A}_{\text{DET}}}\left(\boldsymbol{t}_i, q_i\right) \otimes \bigoplus_{q_1,\cdots,q_k \in Q} \bigotimes_{i=1}^{k} r_{\mathcal{Q}}\left(q_i\right) \otimes \bigoplus_{\langle q_1,\cdots,q_k\rangle \xrightarrow{a/w} q'} w \tag{1.40}$$

$$= \bigotimes_{i=1}^{k} \boldsymbol{\alpha}_{\mathcal{A}_{\text{DET}}}\left(\boldsymbol{t}_i, q_i\right) \otimes w' \otimes w'^{-1} \otimes \bigoplus_{q_1,\cdots,q_k \in Q} \bigotimes_{i=1}^{k} r_{\mathcal{Q}}\left(q_i\right) \otimes \bigoplus_{\langle q_1,\cdots,q_k\rangle \xrightarrow{a/w} q'} w \tag{1.41}$$

$$= \left(\bigotimes_{i=1}^{k} \boldsymbol{\alpha}_{\mathcal{A}_{\text{DET}}}\left(\boldsymbol{t}_i, q_i\right) \otimes w'\right) \otimes \left(\bigoplus_{q_1,\cdots,q_k \in Q} w'^{-1} \otimes \bigotimes_{i=1}^{k} r_{\mathcal{Q}}\left(q_i\right) \otimes \bigoplus_{\langle q_1,\cdots,q_k\rangle \xrightarrow{a/w} q'} w\right) \tag{1.42}$$

$$= \left(\bigotimes_{i=1}^{k} \boldsymbol{\alpha}_{\mathcal{A}_{\text{DET}}}\left(\boldsymbol{t}_i, q_i\right) \otimes w'\right) \otimes \left(\bigoplus_{q_1,\cdots,q_k \in Q} \bigoplus_{\langle q_1,\cdots,q_k\rangle \xrightarrow{a/w} q'} w'^{-1} \otimes \bigotimes_{i=1}^{k} r_{\mathcal{Q}}\left(q_i\right) \otimes w\right) \tag{1.43}$$

$$= \left(\bigotimes_{i=1}^{k} \boldsymbol{\alpha}_{\mathcal{A}_{\text{DET}}}\left(\boldsymbol{t}_i, q_i\right) \otimes w'\right) \otimes r_{\mathcal{Q}}\left(q'\right) \tag{1.44}$$

$$= \boldsymbol{\alpha}_{\mathcal{A}_{\text{DET}}}\left(\boldsymbol{t}, q'\right) \otimes r_{\mathcal{Q}}\left(q'\right) \tag{1.45}$$

This completes the proof. ■

Theorem 1.8.2. *Let $\mathcal{A}$ be an FSTA over the divisible semiring $\mathcal{W}$ and $\mathcal{A}_{\text{DET}}$ the output of running* WeightedDeterminize *on $\mathcal{A}$. Then any tree $\boldsymbol{t}$ has equal tree weights in $\mathcal{A}$ and $\mathcal{A}_{\text{DET}}$.*

Proof.

$$\mathcal{A}(\boldsymbol{t}) = \bigoplus_{q \in Q} \boldsymbol{\alpha}_{\mathcal{A}}\left(\boldsymbol{t}, q\right) \otimes \rho(q) \tag{1.46}$$

$$= \bigoplus_{q \in Q} \boldsymbol{\alpha}_{\mathcal{A}_{\text{DET}}}\left(\boldsymbol{t}, \mathcal{Q}\right) \otimes r_{\mathcal{Q}}\left(q\right) \otimes \rho(q) \tag{1.47}$$

$$= \boldsymbol{\alpha}_{\mathcal{A}_{\text{DET}}}\left(\boldsymbol{t}, \mathcal{Q}\right) \otimes \left(\bigoplus_{q \in Q} r_{\mathcal{Q}}\left(q\right) \otimes \rho(q)\right) \tag{1.48}$$

$$= \boldsymbol{\alpha}_{\mathcal{A}_{\text{DET}}}\left(\boldsymbol{t}, \mathcal{Q}\right) \otimes \rho\left(\mathcal{Q}\right) \tag{1.49}$$

$$= \mathcal{A}_{\text{DET}}(\boldsymbol{t}) \tag{1.50}$$

■

1.9 Unweighted Minimization

Before diving into the algorithms and proofs, we need to present a new type of context in addition to the one from Def. 1.3.1. Recall that the context defined in Def. 1.3.1 was a just a tree with a gap at one of its leaves. Here, we define a new type of context, over a set of states Q and a ranked alphabet Σ, which resembles a path through an FSTA.

Definition 1.9.1 (Context)**.** *The set $\mathcal{C}_\Sigma(Q)$ of **contexts** defined over the set of states Q and the ranked alphabet Σ is the smallest set defined inductively as follows:*

- $\square \subseteq \mathcal{C}_\Sigma(Q)$*, where $\square$ is a special symbol that does not belong to the alphabet,*
- *If $\boldsymbol{t}_1, \boldsymbol{t}_2, \ldots, \boldsymbol{t}_{i-1}, \boldsymbol{t}_{i+1}, \ldots, \boldsymbol{t}_k \in \mathcal{T}_\Sigma(Q)$ for $i = 1, \cdots, k$, $\mathbf{c} \in \mathcal{C}_\Sigma(Q)$, $q \in Q$ and $a \in \Sigma^k$ then $\langle \boldsymbol{t}_1, \boldsymbol{t}_2, \ldots, \boldsymbol{t}_{i-1}, \mathbf{c}, \boldsymbol{t}_{i+1}, \ldots, \boldsymbol{t}_k \rangle \xrightarrow{a} q \in \mathcal{C}_\Sigma(Q)$.*

Definition 1.9.2. *Let $\mathcal{A} = (\Sigma, Q, I, F, \delta)$ be a finite-state tree automaton. The FSTA $\mathcal{A}'$ is a **minimal automaton** with regards to $\mathcal{A}$ if it is equivalent to $\mathcal{A}$ and has the lowest number of states among all FSTAs that are equivalent to $\mathcal{A}$.*

Definition 1.9.3 (Equivalence of states)**.** *Let $\mathcal{A} = (\Sigma, Q, I, F, \delta)$ be an unweighted FSTA. Two states $q, q' \in Q$ are **equivalent**, denoted with $q \sim_\delta q'$ if for any context $\mathbf{c} \in \mathcal{C}_\Sigma(Q)$, $\delta(\mathbf{c}[q]) \in F \iff \delta(\mathbf{c}[q']) \in F$. If the states are not equivalent, we call them **distinguishable**.*

Example 1.9.1. *Consider the ranked alphabet $\Sigma = \{f:2, g:1, a:0, b:0, c:0, d:0\}$. Example 1.9.1 shows 2 FSTAs, $\mathcal{A}$ and $\mathcal{A}'$ that recognize the language $L = \{g(f(a,b)), g(f(a,d)), g(f(c,b)), g(f(c,d))\}$ over Σ. The automaton $\mathcal{A}'$ is the minimal automaton that recognizes L. The tree from Fig. 1.12b is an example of a context $\mathbf{c} \in \mathcal{C}_\Sigma(Q)$, resulted by replacing a leaf node of a hyperpath of $\mathcal{A}$ with the gap $\square$. In $\mathcal{A}$, states 1 and 3 are equivalent since it holds that $\forall \mathbf{c}, \delta(\mathbf{c}[1]) = \delta(\mathbf{c}[3]) \in F$. The context from Fig. 1.12b is an example of such context. All the other pairs of states are distinguishable.*

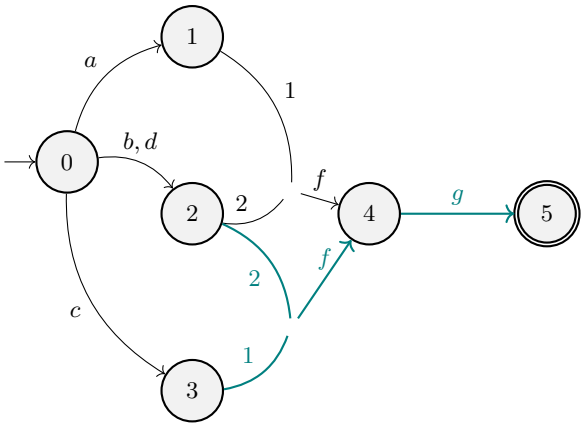
Theorem 1.9.1. *The equivalence of states as defined in Def. 1.9.3 is an equivalence relation.*

Proof. In order to prove the theorem, we need to show reflexivity, symmetry and transitivity. The first two properties are left as exercises, while the transitivity property can be proved as in the case of string automata. Let $\mathcal{A} = (\Sigma, Q, I, F, \delta)$ be a FSTA and q, q', q'' states in Q. By contradiction, assume that $\sim_\delta$ is not transitive: $q \sim_\delta q'$ and $q' \sim_\delta q''$ but q and q'' are distinguishable. Then there must exist some context $\mathbf{c} \in \mathcal{C}_\Sigma(Q)$ that distinguishes them, i.e. either $\delta(\mathbf{c}[q]) \in F$ or $\delta(\mathbf{c}[q'']) \in F$. Suppose $\delta(\mathbf{c}[q])$ is final. If $\delta(\mathbf{c}[q'])$ is final, then q' and q'' are distinguishable, which leads to a contradiction. If $\delta(\mathbf{c}[q'])$ is not final, q and q' are distinguishable, which again leads to a contradiction. We can use the same arguments for the case when $\delta(\mathbf{c}[q''])$ is final. This means that the relation $\sim_\delta$ must be an equivalence relation. ■

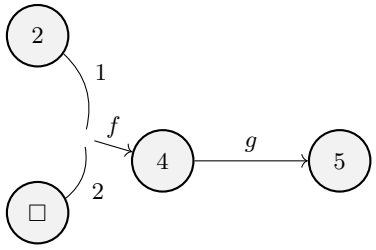
Definition 1.9.4 (Nerode congruence)**.** *Let Σ be a ranked alphabet, L a language over Σ and $\boldsymbol{t}, \boldsymbol{t}' \in \mathcal{T}(\Sigma)$ trees over Σ. We define the relation $\sim_L$ on all trees over some ranked alphabet as $\boldsymbol{t} \sim_L \boldsymbol{t}'$ if and only if for all contexts $\mathbf{c} \in \mathcal{C}(\Sigma)$, $\mathbf{c}[\boldsymbol{t}] \in L \iff \mathbf{c}[\boldsymbol{t}'] \in L$. We call the relation $\sim_L$ the **Nerode congruence**.*

Example 1.9.2. *Recall the language $L = \{g(f(a,b)), g(f(a,d)), g(f(c,b)), g(f(c,d))\}$ recognized by the automaton $\mathcal{A}$ from Example 1.9.1. For the trees $\boldsymbol{t} = a$ and $\boldsymbol{t} = c$ it holds that $\forall \mathbf{c} \in \mathcal{C}(\Sigma), \mathbf{c}[\boldsymbol{t}] \in L \iff \mathbf{c}[\boldsymbol{t}'] \in L$, therefore $\boldsymbol{t} \sim_L \boldsymbol{t}'$. For any other context $\mathbf{c}'$ it holds that $\mathbf{c}'[a], \mathbf{c}'[c] \notin L$.*

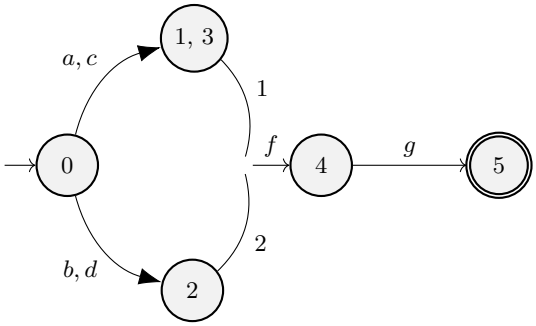
Theorem 1.9.2 (Myhill-Nerode 1)**.** *If a tree language L is regular, $\sim_L$ has a finite number of equivalence classes.*



(a) Example of a FSTA $\mathcal{A}$.



(b) Example of a context $\mathbf{c} \in \mathcal{C}_\Sigma(Q)$.



(c) Minimal FSTA $\mathcal{A}'$.

Figure 1.12: Example of FSTAs $\mathcal{A}$ and $\mathcal{A}'$ over the alphabet Σ that recognize the language L, together with an example of a context $\mathbf{c} \in \mathcal{C}_\Sigma(Q)$. $\mathcal{A}'$ is the minimal automaton that recognizes L. The numbers on the hyperedges indicate the index of the state on the left-hand side of the transition.

Figure 1.13: Example of contexts $\mathbf{c} \in \mathcal{C}(\Sigma)$ such that $\mathbf{c}[a] \in L \iff \mathbf{c}[c] \in L$, therefore $a \sim_L c$.

Proof. If L is a regular tree language, by definition there must exist a deterministic FSTA $\mathcal{A} = (\Sigma, Q, I, F, \delta)$ that recognizes it. Consider the subsets $P_m, m \in [1, |Q|]$ containing trees $\boldsymbol{t} \in \mathcal{T}(\Sigma)$ (not necessarily in L) such that, when input to $\mathcal{A}$, lead to state q_m. There exists a single state q_m for any tree as $\mathcal{A}$ is deterministic. Then for any two trees $\boldsymbol{t}, \boldsymbol{t}' \in P_m$ and any context $\mathbf{c} \in \mathcal{C}(\Sigma)$, it holds that $n(\mathbf{c}[\boldsymbol{t}]) = n(\mathbf{c}[\boldsymbol{t}'])$, i.e., when input to $\mathcal{A}$ they lead to the same state. Therefore each block P_m is contained in an equivalence class of $\sim_L$. Additionally, each tree $\boldsymbol{t} \in L$ is contained in an element of some block P_m. Therefore, there must be a many-to-one relation from Q to the equivalence classes of $\sim_L$. This means that the number of equivalence classes must be bounded by $|Q|$, which by definition is finite. Moreover, L is the union of the subsets P_m such that q_m is a final

state. ■

Theorem 1.9.3 (Myhill-Nerode 2)**.** *Let L be some tree language. If $\sim_L$ has a finite number of equivalence classes, then L is regular.*

Proof. Assume $\sim_L$ has a finite number of equivalence classes for the language $L \in \mathcal{T}(\Sigma)$. We denote the equivalence classes by $[\boldsymbol{t}]$, where $\boldsymbol{t} \in L$ is a tree chosen to be representative of the class. Then we can show that we can construct a deterministic FSTA $\mathcal{A} = (\Sigma, Q, q_I, F, \delta)$ that accepts L. For every equivalence class $[\boldsymbol{t}]$, create a state $q_{[\boldsymbol{t}]}$. The initial state q_I is the state $q_{[\varepsilon]}$ associated with the equivalence class of the empty tree ε. For any trees $\boldsymbol{t}_1, \cdots, \boldsymbol{t}_k, \boldsymbol{t}'$ such that $\boldsymbol{t}' = a\langle \boldsymbol{t}_1, \cdots, \boldsymbol{t}_k \rangle$ and $a \in \Sigma$, add a transition $\langle q_{[\boldsymbol{t}_1]}, \cdots, q_{[\boldsymbol{t}_k]} \rangle \xrightarrow{a} q_{[\boldsymbol{t}']}$. Finally, if the tree $\boldsymbol{t}'$ is in the language, add $q_{[\boldsymbol{t}']}$ to the set of final states F. ■

Theorem 1.9.4 (Myhill-Nerode 3)**.** *Let L be a tree language such that $\sim_L$ has a finite number of equivalence classes. Then the FSTA $\mathcal{A}_{\text{MIN}} = (\Sigma_{\text{MIN}}, Q_{\text{MIN}}, I_{\text{MIN}}, F_{\text{MIN}}, \delta_{\text{MIN}})$ constructed as in the proof of Thm. 1.9.3 is the minimal automaton accepting L and is unique up to the renaming of states.*

Proof. Suppose there exists an automaton $\mathcal{A}' = (\Sigma, Q', {q_I}', F', \delta')$ recognizing L that has fewer states than $\mathcal{A}_{\text{MIN}}$. Both $\mathcal{A}'$ and $\mathcal{A}_{\text{MIN}}$ are trim as otherwise the machines obtained by trimming them would have fewer states, which would lead to a contradiction. As in the proof for finite-state automata, we construct the union of the two machines $\mathcal{U} = \mathcal{A}_{\text{MIN}} \cup \mathcal{A}'$ without adding any new states. A state in $\mathcal{U}$ is final if it is final in either $\mathcal{A}_{\text{MIN}}$ or $\mathcal{A}'$. Additionally, the initial states of $\mathcal{A}_{\text{MIN}}$ and $\mathcal{A}'$ are equivalent since the machines are equivalent. When reading a tree that leads to a pair of equivalent states $q \in Q_{\text{MIN}}, q' \in Q'$, all intermediate states resulted from reading any subtree must also be equivalent. This can easily be proven by induction on the structure of the tree. When the tree is a leaf a, since the initial states are equivalent, the states q, q' reached by the transitions $\langle q_{I\,\text{MIN}} \rangle \xrightarrow{a} q$ and $\langle {q_I}' \rangle \xrightarrow{a} q'$ must also be equivalent. Now let $[q_1] = [q'_1], \cdots, [q_k] = [q'_k]$ be the equivalent states reached by reading the subtrees $\boldsymbol{t}_1, \cdots, \boldsymbol{t}_k$ of the tree $\boldsymbol{t} = a\langle \boldsymbol{t}_1, \cdots, \boldsymbol{t}_k \rangle$. Then the states q, q' reached by the transitions $\langle q_1, \cdots, q_k \rangle \xrightarrow{a} q$ and $\langle q'_1, \cdots, q'_k \rangle \xrightarrow{a} q'$ must also be equivalent.

This implies that every state $q \in Q_{\text{MIN}}$ is equivalent to at least one state $q' \in Q'$. By the pigeonhole principle, since $\mathcal{A}'$ has fewer states than $\mathcal{A}_{\text{MIN}}$, there must be at least two states $q_1, q_2 \in Q_{\text{MIN}}$ that are equivalent to some state $q' \in Q'$. By transitivity, this means that q_1 and q_2 are equivalent to themselves. This is obviously a contradiction as they could be merged and result in an automaton with fewer states. Thus, $\mathcal{A}_{\text{MIN}}$ must have the minimal number of states.

The uniqueness of $\mathcal{A}_{\text{MIN}}$ can be proven in a similar way. Let $\mathcal{A}_1 = (\Sigma_1, Q_1, I_1, F_1, \delta_1)$ and $\mathcal{A}_2 = (\Sigma_2, Q_2, I_2, F_2, \delta_2)$ be two minimal FSTAs accepting L. For any state $q_1 \in Q_1$, there cannot be more than one equivalent state in $q_2 \in Q_2$, otherwise these could be merged. Therefore there must be a one-to-one mapping between the states of $\mathcal{A}_1$ and $\mathcal{A}_2$. This implies that the minimal automata accepting some language L are unique up to the renaming of states. ■

FSTA Minimization as Partition Refinement

Perhaps not surprisingly, the partition refinement machinery that we developed for finite-state automata can be reused for performing FSTA minimization. In fact, there is only a small change that we need to make to the FSA algorithm in order to obtain a Hopcroft-style minimization algorithm for unweighted FSTAs. In the FSA case, the minimization algorithm consisted of multiple runs of partition refinement, once per each input symbol. For tree automata the partition refinement must be run once for every *pair of symbols and contexts.*

Let us explain in more detail the intuition behind this. Recall that we have the following implication for all pairs of equivalent states $q_1 \sim_\delta q'_1$, $\cdots$, $q_k \sim_\delta q'_k$ and symbols $a \in \Sigma^k$:

$$\forall q_i \sim_\delta q'_i, i \in [1, k] \Rightarrow \delta\left(\langle q_1, \cdots, q_k \rangle, a\right) \sim_\delta \delta\left(\langle q'_1, \cdots, q'_k \rangle, a\right) \tag{1.51}$$

where $\delta\left(\langle q_1, \cdots, q_k \rangle, a\right)$ denotes the state reached by following a transition labeled with a from the source states $\langle q_1, \cdots, q_k \rangle$.

Now let $\mathcal{A} = (\Sigma, Q, I, F, \delta)$ be an unweighted FSTA. From each transition $\langle q_1, \cdots, q, \cdots, q_k \rangle \xrightarrow{a} p$ in δ we can construct the context $\langle q_1, \cdots, \square, \cdots, q_k \rangle \xrightarrow{a} p$ for the state q. We iteratively partition the states according to the transition function restricted to a certain $a \in \Sigma$ and a context. In other words, we can construct the function f that maps a state q to its target state q' in the transition corresponding to the context derived as explained previously. The state q' must be unique as the automaton $\mathcal{A}$ is assumed to be deterministic. This intuition is formalized in Alg. 4.

Algorithm 4 The FSTA PartitionRefinementMinimization algorithm. partition can be any partition refinement algorithm.

```
1. def PartitionRefinementMinimization(𝒜):
2.   𝒫 ← {F, Q \ F}                                   ▷ Initialize the partition.
3.   for ⟨q_1, ..., q_k⟩ --a/•--> p ∈ δ :
4.     for i = 1, ..., k :
5.       ▷ Symbol–context-specific state mapping
6.       f ← { q_i ↦ p | ⟨q_1, ..., q_{i-1}, q_i, q_{i+1}, ..., q_k⟩ --a/•--> p ∈ δ }
7.       𝒫 ← partition(f, 𝒫)
8.   return BlockFSTAConstruction(𝒜, 𝒫)
```

Complexity The runtime of this algorithm highly depends on the algorithm used for partition refinement. In particular, if we use Hopcroft as the partition function, we get a complexity of $\mathcal{O}(k|\Sigma||Q|\log(|Q|))$, where k is the maximum arity of the symbols in Σ.

Now that we explained how to partition Q into subsets of equivalent states, we need to show how to actually construct the minimal automaton equivalent to some FSTA $\mathcal{A}$. Alg. 5 shows formally how to do this. As we will reuse the same algorithm for weighted minimization, we present a weighted version of the algorithm and use the Boolean semiring in the unweighted case. The algorithm receives as input a pre-computed partition $\mathcal{P}$ of the states into their equivalence classes. The states Q_{MIN} of the minimal automaton represent the subsets of $\mathcal{P}$. Then for each transition in the original machine $\mathcal{A}$, we add a transition in $\mathcal{A}_{\text{MIN}}$ where we replace each state q with the state corresponding to its equivalence class $q_{[q]} \in Q_{\text{MIN}}$. If the same transition is added multiple times, their weights are $\oplus$-summed. Additionally, we $\oplus$-sum the weights of the intial and final states.

Algorithm 5 The BlockFSTAConstruction algorithm.

def BlockFSTAConstruction$(\mathcal{A}, \mathcal{P})$:
  $\mathcal{A}_{\text{MIN}} \leftarrow (\Sigma, Q_{\text{MIN}}, I_{\text{MIN}}, F_{\text{MIN}}, \delta_{\text{MIN}})$ ▷*Initialize a new automaton over the same semiring as* $\mathcal{A}$.
  $\mu \leftarrow \{\}$ ▷*Initialize an empty map for bookkeeping.*
  for $\mathcal{Q} \in \mathcal{P}$:
    for $q \in \mathcal{Q}$:
      $\mu[q] \leftarrow \mathcal{Q}$ ▷*Link each state* q *to its equivalence class.*
  for $\langle q_1, \ldots, q_k \rangle \xrightarrow{a/w} q' \in \delta$:
    add transition $\langle \mu[q_1], \ldots, \mu[q_k] \rangle \xrightarrow{a/w} \mu[q']$ to $\mathcal{A}_{\text{MIN}}$
  for $q_I \in I$:
    $\lambda_{\text{MIN}}(\mu[q_I]) \leftarrow \lambda_{\text{MIN}}(\mu[q_I]) \oplus \lambda(q_I)$
  for $q_F \in F$:
    $\rho_{\text{MIN}}(\mu[q_F]) \leftarrow \rho_{\text{MIN}}(\mu[q_F]) \oplus \rho(q_F)$
  return $\mathcal{A}_{\text{MIN}}$

1.10 Weight Pushing

Just like in the case of finite-state automata, weight pushing for bottom-up tree automata seeks to manipulate the weights of the transitions, initial and final states such that the resulting weighting of the automaton is canonical. In a finite-state string automaton the weights are *pushed* along the path from the final state to the initial state. Analogously, in a bottom-up automaton the weights are moved from the root towards the leaves. The obtained canonical automaton can then be minimized using unweighted minimization algorithms, treating the symbol and the weight as a single transition label.

Let $\mathcal{A} = (\Sigma, Q, \lambda, \rho, \delta)$ be a weighted bottom-up finite-state tree automaton defined over the *commutative semifield* $\mathcal{W} = (\mathbb{K}, \oplus, \otimes, \mathbf{0}, \mathbf{1})$. The automaton $\mathcal{A}$ must also be *deterministic* and *complete*. The redistribution of the weights will be done according to some *potential function* $\Psi : Q \to \mathbb{K}$ such that $\Psi(q) \neq \mathbf{0}, \forall q \in Q$. The weights in the pushed automaton are defined as follows:

$$\forall q \in Q, \qquad \lambda_{\text{PUSH}}(q) \leftarrow \lambda(q) \otimes \Psi(q) \tag{1.52}$$

$$\forall e = \langle q_1, q_2, \ldots, q_k \rangle \xrightarrow{a/w} q' \in \delta, \qquad w_{\text{PUSH}}(e) \leftarrow \bigotimes_{i=1}^{k} \Psi(q_i)^{-1} \otimes w \otimes \Psi(q') \tag{1.53}$$

$$\forall q \in Q, \qquad \rho_{\text{PUSH}}(q) \leftarrow \Psi(q)^{-1} \otimes \rho(q). \tag{1.54}$$

Now we have to show how to actually compute the potentials for each state $q \in Q$. The algorithm is based on Hanneforth et al. (2017) and is shown in Algorithm 6. The algorithm requires an already-computed coarsest partition of the set of states into their equivalence classes. This can be done as presented the previous section focused on unweighted minimization.

At an intuitive level, the algorithm computes the potential weights and a path leading to a final state (context) for each state, starting from the final states and moving to the initial states in a breadth-first search manner. By exploring the states by their equivalence classes, it ensures that the weights of equivalent states receive the same context. This allows for a proper canonicalization which means that after pushing, equivalent states have equally weighted corresponding transitions,

i.e. $\langle q_1, \cdots, q_k \rangle \xrightarrow{a/w} p$ and $\langle q'_1, \cdots, q'_k \rangle \xrightarrow{a/w} p'$ have equal weights if $[q_i] = [q'_i], \forall i \in [1, k], [p] = [p']$. This in turn means that all paths to the final states will be weighted equally for equivalent states. This is analogous to having equal backward weights in the case of WFSTAs.

At initialization, since all final states are trivially co-accessible, their contexts are set to the trivial context and their potential weights are set to their final weights. Additionally, all transitions leading to the final states are added to the queue. Then, for each transition $\langle q_1, q_2, \ldots, q_k \rangle \xrightarrow{\bullet/\bullet} \bullet$ that we take from the queue we check whether any of the source states q_i are not marked as co-accessible yet. We mark all the states in q_i's block as co-accessible and set their contexts to the context of the target state (which we already computed as otherwise the transition would not have been in queue) in which we insert the context created from the current transition. To compute the potential weights for each state q in the current block, we find its corresponding transition by replacing q_i with q in the left-hand side of the current transition and $\otimes$-multiply its weight with the potential of the target state. All transitions leading to newly explored nodes are added to queue. The algorithm finishes when the queue as been emptied, at which point the potential weights of all co-accessible states have been computed. The other states are non co-accessible and we do not compute their weights as they cannot be on any path from the initial states to the final states.

Algorithm 6 The Quasi-Backwards algorithm.

```
1.  def Quasi-Backwards(A, P):
2.    queue ← ∅
3.    C ← F                                   ▷All final states are trivially co-accessible.
4.    for q ∈ F :
5.      Ψ (q) ← ρ (q)
6.      path (q) ← □                          ▷Path to final states is trivial context.
7.      for • --•/•--> q ∈ δ :
8.        push transition • --•/•--> q onto queue   ▷Add to queue all transitions to final states.
9.    while |queue| > 0 :
10.     pop ⟨q_1, ..., q_k⟩ --a/•--> q' from queue
11.     for i = 1, ··· , k :
12.       if q_i ∉ C :
13.         C ← C ∪ [q_i]
14.         let c = ⟨q_1, ..., q_{i-1}, □, q_{i+1}, ..., q_k⟩ --a--> q'   ▷Prepare context.
15.         for p ∈ [q_i] :
16.           ▷This loop only iterates once as A is deterministic.
17.           for ⟨q_1, ..., q_{i-1}, p, q_{i+1}, ..., q_k⟩ --a/w--> s in δ :
18.             Ψ (p) ← w ⊗ Ψ (s)
19.             path (p) ← path (q') [c]
20.           for • --•/•--> p ∈ δ :
21.             push transition • --•/•--> p onto queue
22.   return Ψ, path
```

Complexity In the first *for loop* we set the context of each final state to the trivial context and their potentials to $\mathbf{1}$. This is done once for each state, thus this initialization takes time $\mathcal{O}(|Q|)$. Additionally, we push onto the stack all transitions leading to final states, which takes $\mathcal{O}(|\delta|)$. Each

Algorithm 7 The WFSTA WeightPush algorithm. We assume that we have an already-computed partition of the states $\mathcal{P}$.

1. **def** WeightPush$(\mathcal{A}, \mathcal{P})$:
2. $\quad \Psi \leftarrow$ Quasi-Backwards$(\mathcal{A}, \mathcal{P})$
3. $\quad \triangleright$ *Initialize the automaton* $\mathcal{A}_{\text{PUSH}}$ *over the same semiring as* $\mathcal{A}$
4. $\quad \mathcal{A}_{\text{PUSH}} \leftarrow (\Sigma, Q_{\text{PUSH}}, \lambda_{\text{PUSH}}, \rho_{\text{PUSH}}, \delta_{\text{PUSH}})$
5. $\quad$ **for** $q \in Q$:
6. $\quad\quad \lambda_{\text{PUSH}}(q) \leftarrow \lambda(q) \otimes \Psi(q)$
7. $\quad\quad \rho_{\text{PUSH}}(q) \leftarrow \Psi(q)^{-1} \otimes \rho(q)$
8. $\quad$ **for** $\langle q_1, \ldots, q_k \rangle \xrightarrow{a/w} p \in \delta$:
9. $\quad\quad w' \leftarrow \bigotimes_{i=1}^{k} \Psi(q_i)^{-1} \otimes w \otimes \Psi(p)$
10. $\quad\quad$ add transition $\langle q_1, \ldots, q_k \rangle \xrightarrow{a/w'} p$ to δ_{PUSH}
11. $\quad$ **return** $\mathcal{A}_{\text{PUSH}}$

transition can be pushed and then popped at most once from the queue so the *while loop* can be executed at most once per transition, leading to $\mathcal{O}(|\delta|)$ complexity. Inside the *while loop* we take a transition from queue and look for an unexplored state among the source states. Once we find such state, we compute its context and potential then mark it as co-accessible. This can be done at most once for each state, as we in later iterations we ignore the already explored states. This gives a complexity of $\mathcal{O}(|\delta|)$. In total we obtain the complexity $\mathcal{O}(|Q| + |\delta|)$.

Example 1.10.1. *Figure 1.14 shows an example of WFSTA, together with an equivalent version after weight pushing. We assume that the automaton is complete but only show the relevant transitions. The transitions leading to the sink state were omitted.*

We can now prove that weight preserves the semantics of the original machine, i.e. the tree sum of any tree remains the same after weight pushing. Intuitively this seems correct as every transition to a state $q \in Q$ will charge an additional weight of $\Psi(q)$ and every transition leaving from q will compensate by charging the weight $\Psi(q)^{-1}$. We will start by proving the following lemma.

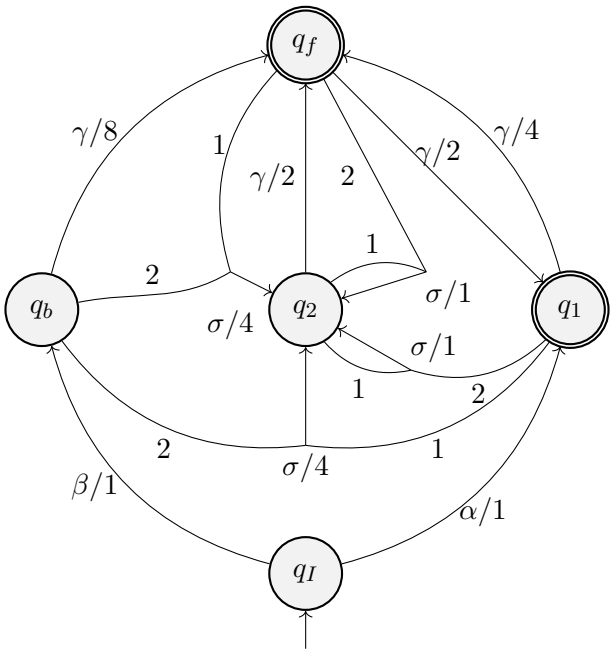
Theorem 1.10.1. *Let* $\mathcal{A}_{\text{PUSH}} = (\Sigma, Q_{\text{PUSH}}, \lambda_{\text{PUSH}}, \rho_{\text{PUSH}}, \delta_{\text{PUSH}})$ *be the result of WeightPush applied to the automaton* $\mathcal{A} = (\Sigma, Q, \lambda, \rho, \delta)$*. Then it holds that*

$$\boldsymbol{\alpha}_{\mathcal{A}_{\text{PUSH}}}(\boldsymbol{t}, q) = \boldsymbol{\alpha}_{\mathcal{A}}(\boldsymbol{t}, q) \otimes \Psi(q) \tag{1.55}$$

for any $\boldsymbol{t} \in T(\Sigma)$.

Proof. We proceed by induction over the structure of the tree. For the base case consider trees $\boldsymbol{t}$ made of a single node such that $\boldsymbol{t} = a, a \in \Sigma^0$.

(a) An example of a bottom-up weighted finite-state automaton.



(b) Bottom-up weighted finite-state automaton after pushing.

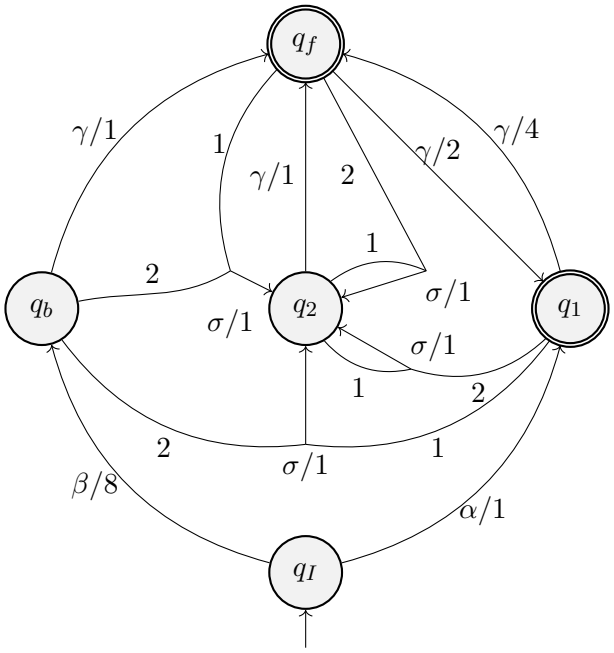


Figure 1.14: Example of bottom-up weighted finite-state tree automaton before and after pushing. Adapted from Hanneforth et al. (2017)

$$\boldsymbol{\alpha}_{\mathcal{A}_{\text{PUSH}}}(a,q) = \bigoplus_{p \xrightarrow{a/w_{\text{PUSH}}} q, p \in I} \lambda_{\text{PUSH}}(p) \otimes w_{\text{PUSH}} \quad \text{(definition)} \tag{1.56}$$

$$= \bigoplus_{p \xrightarrow{a/w_{\text{PUSH}}} q, p \in I} \lambda(p) \otimes \Psi(p) \otimes w_{\text{PUSH}} \quad \text{(definition)} \tag{1.57}$$

$$= \bigoplus_{p \xrightarrow{a/w} q, p \in I} \lambda(p) \otimes \Psi(p) \otimes \Psi(p)^{-1} \otimes w \otimes \Psi(q) \quad \text{(definition)} \tag{1.58}$$

$$= \bigoplus_{p \xrightarrow{a/w} q, p \in I} \lambda(p) \otimes w \otimes \Psi(q) \quad \text{(inverse)} \tag{1.59}$$

$$= \boldsymbol{\alpha}_{\mathcal{A}}(a,q) \otimes \Psi(q) \quad \text{(definition)} \tag{1.60}$$

Now suppose that the tree $\boldsymbol{t}$ has the form $\boldsymbol{t} = a\langle \boldsymbol{t}_1, \boldsymbol{t}_2, \ldots, \boldsymbol{t}_k \rangle$ where $\boldsymbol{t}_1$, $\boldsymbol{t}_2$, $\ldots$, $\boldsymbol{t}_k$ are subtrees for which it holds that $\boldsymbol{\alpha}_{\mathcal{A}_{\text{PUSH}}}(\boldsymbol{t}_i, q_i) = \boldsymbol{\alpha}_{\mathcal{A}}(\boldsymbol{t}_i, q_i) \otimes \Psi(q_i), i \in [1,k]$ and $a \in \Sigma^k$.

$$\boldsymbol{\alpha}_{\mathcal{A}_{\text{PUSH}}}(\boldsymbol{t},q) = \bigoplus_{\langle q_1,q_2,\ldots,q_k\rangle \xrightarrow{a/w_{\text{PUSH}}} q} \bigotimes_{i=1}^{k} \boldsymbol{\alpha}_{\mathcal{A}_{\text{PUSH}}}(\boldsymbol{t}_i, q_i) \otimes w_{\text{PUSH}} \quad \text{(definition)} \tag{1.61}$$

$$= \bigoplus_{\langle q_1,q_2,\ldots,q_k\rangle \xrightarrow{a/w_{\text{PUSH}}} q} \bigotimes_{i=1}^{k} \boldsymbol{\alpha}_{\mathcal{A}}(\boldsymbol{t}_i, q_i) \otimes \Psi(q_i) \otimes w_{\text{PUSH}} \quad \text{(inductive hypothesis)} \tag{1.62}$$

$$= \bigoplus_{\langle q_1,q_2,\ldots,q_k\rangle \xrightarrow{a/w_{\text{PUSH}}} q} \bigotimes_{i=1}^{k} \boldsymbol{\alpha}_{\mathcal{A}}(\boldsymbol{t}_i, q_i) \otimes \bigotimes_{i=1}^{k} \Psi(q_i) \otimes w_{\text{PUSH}} \quad \text{(manipulation)} \tag{1.63}$$

$$= \bigoplus_{\langle q_1,q_2,\ldots,q_k\rangle \xrightarrow{a/w} q} \bigotimes_{i=1}^{k} \boldsymbol{\alpha}_{\mathcal{A}}(\boldsymbol{t}_i, q_i) \otimes \bigotimes_{i=1}^{k} \Psi(q_i) \otimes \bigotimes_{i=1}^{k} \Psi(q_i)^{-1} \otimes w \otimes \Psi(q) \quad \text{(definition)} \tag{1.64}$$

$$= \bigoplus_{\langle q_1,q_2,\ldots,q_k\rangle \xrightarrow{a/w} q} \bigotimes_{i=1}^{k} \boldsymbol{\alpha}_{\mathcal{A}}(\boldsymbol{t}_i, q_i) \otimes w \otimes \Psi(q) \quad \text{(inverse)} \tag{1.65}$$

$$= \boldsymbol{\alpha}_{\mathcal{A}}(\boldsymbol{t},q) \otimes \Psi(q) \quad \text{(definition)} \tag{1.66}$$

■

Theorem 1.10.2. *Let $\mathcal{A}_{\text{PUSH}} = (\Sigma, Q_{\text{PUSH}}, \lambda_{\text{PUSH}}, \rho_{\text{PUSH}}, \delta_{\text{PUSH}})$ be the result of WeightPush applied to the automaton $\mathcal{A} = (\Sigma, Q, \lambda, \rho, \delta)$. The tree sum of any tree $\boldsymbol{t}$ in $\mathcal{A}$ is equal to the tree sum of $\boldsymbol{t}$ in $\mathcal{A}_{\text{PUSH}}$.*

Proof. Note that the topology of $\mathcal{A}$ remains unchanged under pushing, only the initial, final and transition weights are redistributed. We have to prove that for any $\boldsymbol{t} \in T(\Sigma)$ it holds that $\mathcal{A}(\boldsymbol{t}) = \mathcal{A}_{\text{PUSH}}(\boldsymbol{t})$. Recall that the tree sum of a tree $\boldsymbol{t}$ is defined as

$$\mathcal{A}(\boldsymbol{t}) \stackrel{\text{def}}{=} \bigoplus_{q \in F} \boldsymbol{\alpha}_{\mathcal{A}}(\boldsymbol{t}, q) \otimes \rho(q). \tag{1.67}$$

We can plug in the identity we have just proven into the definition of tree sum:

$$\begin{aligned}
\mathcal{A}_{\text{PUSH}}(\boldsymbol{t}) &= \bigoplus_{q \in F} \boldsymbol{\alpha}_{\mathcal{A}_{\text{PUSH}}}(\boldsymbol{t}, q) \otimes \rho_{\text{PUSH}}(q) && \text{(definition)} && (1.68)\\
&= \bigoplus_{q \in F} \boldsymbol{\alpha}_{\mathcal{A}}(\boldsymbol{t}, q) \otimes \Psi(q) \otimes \rho_{\text{PUSH}}(q) && (1.55) && (1.69)\\
&= \bigoplus_{q \in F} \boldsymbol{\alpha}_{\mathcal{A}}(\boldsymbol{t}, q) \otimes \Psi(q) \otimes \Psi(q)^{-1} \otimes \rho(q) && \text{(definition)} && (1.70)\\
&= \bigoplus_{q \in F} \boldsymbol{\alpha}_{\mathcal{A}}(\boldsymbol{t}, q) \otimes \rho(q) && \text{(inverse)} && (1.71)\\
&= \mathcal{A}(\boldsymbol{t}) && \text{(definition)} && (1.72)
\end{aligned}$$

This concludes the proof. ■

Definition 1.10.1 (Tree top)**.** *A **tree top** is a path $\boldsymbol{t} \in \mathcal{T}_{\Sigma}(Q)$ leading to a final state, i.e. $n(\boldsymbol{t}) = q \in F$.*

As the automaton is deterministic, there is a single state that can be reached by following a tree top $\boldsymbol{t}$, thus we can overload $\delta(\boldsymbol{t})$ to denote the final state of $\boldsymbol{t}$. Additionally, we use $w(\boldsymbol{t})$ for the weight of this path. We can define the weight of a tree top similar to the backward weight in an FSA.

Definition 1.10.2. *Let $\mathcal{A} = (\Sigma, Q, \delta, \lambda, \rho)$ be a WFSTA. The **backward weight** $\boldsymbol{\beta}_{\mathcal{A}}(\boldsymbol{t})$ of the tree top $\boldsymbol{t}$ under $\mathcal{A}$ is defined as*

$$\boldsymbol{\beta}_{\mathcal{A}}(\boldsymbol{t}) = w(\boldsymbol{t}) \otimes \rho(n(\boldsymbol{t})) \tag{1.73}$$

where $w(\boldsymbol{t})$ is the weight of the path induced by $\boldsymbol{t}$ and can be defined inductively as follows:

- $w(\boldsymbol{t}) = 1$ *if* $\boldsymbol{t} = q, q \in Q$;
- $w(\boldsymbol{t}) = \bigotimes_{i=1}^{k} w(\boldsymbol{t}_i) \otimes w$ *if* $\boldsymbol{t} = \langle \boldsymbol{t}_1, \cdots, \boldsymbol{t}_k \rangle \xrightarrow{a} q$ *and* $\langle n(\boldsymbol{t}_1), \cdots, n(\boldsymbol{t}_k) \rangle \xrightarrow{a/w} q \in \delta$.

Theorem 1.10.3. *The potential weight $\Psi(q)$ for a state $q \in Q$ computed by the Quasi-Backwards algorithm is equal to the weight of the tree top $\mathsf{path}(q)[q]$, i.e. $\Psi(q) = \boldsymbol{\beta}_{\mathcal{A}}(\mathsf{path}(q)[q])$.*

Proof. We prove the statement by induction along the main loop of the algorithm.

Base Case. Initially, the potential weights of the final states $q \in F$ are set to $\rho(q)$ and their contexts are set to the trivial context $\square$, i.e. $\mathsf{path}(q), q \in F$. Then it holds that $\mathsf{path}(q)[q] = q$, whose backward weight is $\boldsymbol{\beta}_{\mathcal{A}}(q) = \mathbf{1} \otimes \rho(q) = \rho(q)$ by definition.

Inductive Step. Inside the main loop we set $\Psi(q) \leftarrow \Psi(p) \otimes w$, where w is the weight of current transition $\langle \cdots q \cdots \rangle \xrightarrow{a/w} p$. By the inductive hypothesis, it holds that $\Psi(p) = \boldsymbol{\beta}_{\mathcal{A}}(\mathsf{path}(p)[p])$. The equivalence class of p has already been explored in a previous iteration, otherwise the current transition would not be in the queue. Therefore $\Psi(p)$ is already computed.

$$\Psi(q) = \Psi(p) \otimes w\left(\langle \cdots q \cdots \rangle \xrightarrow{a/w} p\right) \tag{1.74}$$

$$= \boldsymbol{\beta}_{\mathcal{A}}(\mathsf{path}(p)[p]) \otimes w\left(\langle \cdots q \cdots \rangle \xrightarrow{a/w} p\right) \tag{1.75}$$

$$= \boldsymbol{\beta}_{\mathcal{A}}(\mathsf{path}(p)[\mathbf{c}[q]]) \tag{1.76}$$

$$= \boldsymbol{\beta}_{\mathcal{A}}(\mathsf{path}(q)[q]) \tag{1.77}$$

In the second line we only multiply with the weight of the current transition as the automaton is deterministic, thus there is a single transition of the form $\langle \cdots q \cdots \rangle \xrightarrow{a/\bullet} \bullet$. Furthermore, we obtain the last line by plugging in the definition of the path for state q, expressed in terms of the path for state p, which was computed in a previous iteration for the whole equivalence class $[p]$. ■

1.11 Weighted Minimization

Following the definitions from Maletti (2008), we define the Nerode congruence for trees in a weighted tree language.

Definition 1.11.1 (Tree equivalence)**.** *Let $\mathcal{A} = (\Sigma, Q, \lambda, \rho, \delta)$ be a WFSTA over the semiring $\mathcal{W} = (\mathbb{K}, \oplus, \otimes, \boldsymbol{0}, \boldsymbol{1})$ and $\boldsymbol{t}, \boldsymbol{t}' \in L(\mathcal{A})$ two trees in $\mathcal{A}$'s language. We say that $\boldsymbol{t}$ and $\boldsymbol{t}'$ are **equivalent**, denoted by $\boldsymbol{t} \sim_L \boldsymbol{t}'$, if and only if there exists a $k \in \mathbb{K}$ and a $k' \in \mathbb{K}$ such that for any context $\mathbf{c} \in \mathcal{C}(\Sigma)$*

$$k^{-1} \otimes \mathcal{A}(\mathbf{c}[\boldsymbol{t}]) = k'^{-1} \otimes \mathcal{A}(\mathbf{c}[\boldsymbol{t}']), \tag{1.78}$$

where $\mathcal{A}(C[\boldsymbol{t}])$ and $\mathcal{A}(\mathbf{c}[\boldsymbol{t}'])$ are the tree sums of $\mathbf{c}[\boldsymbol{t}]$ and $\mathbf{c}[\boldsymbol{t}']$ under $\mathcal{A}$.

Definition 1.11.2 (Equivalence of states)**.** *Let $\mathcal{A} = (\Sigma, Q, \lambda, \rho, \delta)$ be a WFSTA over the semiring $\mathcal{W} = (\mathbb{K}, \oplus, \otimes, \boldsymbol{0}, \boldsymbol{1})$. Two states $q, q' \in Q$ are **equivalent** if there exist factors $k \in \mathbb{K}$ and a $k' \in \mathbb{K}$ such that for any context $\mathbf{c} \in \mathcal{C}_\Sigma(Q)$*

$$k^{-1} \otimes \boldsymbol{\beta}_{\mathcal{A}}(\mathbf{c}[q]) = k'^{-1} \otimes \boldsymbol{\beta}_{\mathcal{A}}(\mathbf{c}[q']). \tag{1.79}$$

We denote the equivalence relation by $\sim_\delta$ and the equivalence class of a state q by $[q]$.

Using these definitions, we are now ready to prove the main theorem that will allow us to do minimization in the weighted case like in the unweighted case. We start by proving the following lemma.

Lemma 1.11.1. *Let $\mathcal{A}$ be a WFSTA such that the states $q_i \sim_\delta q_i'$ and $p_i \sim_\delta p_i'$ in Q are equivalent. After pushing it holds that the weights of the transitions $\langle q_1', \cdots, q_{i-1}', q_i, q_{i+1} \cdots, q_k \rangle \xrightarrow{a/w_{\mathcal{A}_{\text{PUSH}}}} p_i$ and $\langle q_1', \cdots, q_{i-1}', q_i', q_{i+1} \cdots, q_k \rangle \xrightarrow{a/w_{\mathcal{A}_{\text{PUSH}}}} p_i'$ are equal. The transitions have the same left-hand side, except at position i, where q_i is replaced with q_i'.*

Proof. We leave the proof as an exercise. ■

Lemma 1.11.2. *Let $\mathcal{A}$ be a WFSTA such that the states $p \sim_\delta p'$, $q_i \sim_\delta q_i', \forall i \in [1, k]$ in Q are pairwise equivalent. After pushing it holds that the weights of the transitions $\langle q_1, q_2, \ldots, q_k \rangle \xrightarrow{a/w_{\mathcal{A}_{\text{PUSH}}}} p$ and $\langle q_1', q_2', \ldots, q_k' \rangle \xrightarrow{a/w'_{\mathcal{A}_{\text{PUSH}}}} p'$ are equal.*

Proof. Recall that the weights of the transitions after pushing are $w_{\mathcal{A}_{\text{PUSH}}} = \bigotimes_{i=1}^{k} \Psi(q_i)^{-1} \otimes w_{\mathcal{A}} \otimes \Psi(p)$ and $w'_{\mathcal{A}_{\text{PUSH}}} = \bigotimes_{i=1}^{k} \Psi(q_i')^{-1} \otimes w'_{\mathcal{A}} \otimes \Psi(p')$. In order to get the desired statement, we can apply the previous lemma k times:

$$w_{\mathcal{A}_{\text{PUSH}}} = \bigotimes_{i=1}^{k} \Psi\left(q_i\right)^{-1} \otimes w_{\mathcal{A}} \otimes \Psi\left(p\right) \tag{1.80}$$

$$= \bigotimes_{i=1}^{1} \Psi\left(q_i\right)^{-1} \otimes \bigotimes_{i=2}^{k} \Psi\left(q_i\right)^{-1} \otimes w_{\mathcal{A}}\left(\left\langle q_1, q_2, \cdots, q_k\right\rangle \xrightarrow{a/w_{\mathcal{A}}} p_1\right) \otimes \Psi\left(p_1\right) \tag{1.81}$$

$$= \bigotimes_{i=1}^{1} \Psi\left(q_i'\right)^{-1} \otimes \bigotimes_{i=2}^{k} \Psi\left(q_i\right)^{-1} \otimes w_{\mathcal{A}}\left(\left\langle q_1', q_2, \cdots, q_k\right\rangle \xrightarrow{a/w_{\mathcal{A}}} p_1'\right) \otimes \Psi\left(p_1'\right) \tag{1.82}$$

$$= \cdots \tag{1.83}$$

$$= \bigotimes_{i=1}^{k-1} \Psi\left(q_i'\right)^{-1} \otimes \bigotimes_{i=k-1}^{k} \Psi\left(q_i\right)^{-1} \otimes w_{\mathcal{A}}\left(\left\langle q_1', \cdots, q_{k-1}', q_k\right\rangle \xrightarrow{a/w_{\mathcal{A}}} p_k\right) \otimes \Psi\left(p_k\right) \tag{1.84}$$

$$= \bigotimes_{i=1}^{k} \Psi\left(q_i'\right)^{-1} w_{\mathcal{A}}\left(\left\langle q_1', \cdots, q_{k-1}', q_k'\right\rangle \xrightarrow{a/w_{\mathcal{A}}} p'\right) \otimes \Psi\left(p'\right) \tag{1.85}$$

$$= w'_{\mathcal{A}_{\text{PUSH}}} \tag{1.86}$$

■

Theorem 1.11.1. *Let $\mathcal{A} = (\Sigma, Q, \lambda, \rho, \delta)$ and let $\mathcal{A}_{\text{PUSH}} = (\Sigma, Q_{\text{PUSH}}, \lambda_{\text{PUSH}}, \rho_{\text{PUSH}}, \delta_{\text{PUSH}})$ its pushed variant. Then for any two states q, q', $q \sim_\delta q'$ implies the following:*

$$\forall \mathbf{c} \in \mathcal{C}_\Sigma\left(Q\right), n\left(\mathbf{c}[q]\right) \in F, n\left(\mathbf{c}[q']\right) \in F \Longrightarrow \boldsymbol{\beta}_{\mathcal{A}_{\text{PUSH}}}\left(\mathbf{c}[q]\right) = \boldsymbol{\beta}_{\mathcal{A}_{\text{PUSH}}}\left(\mathbf{c}[q']\right). \tag{1.87}$$

Proof. We prove the theorem by induction over the structure of the context $\mathbf{c}$. **Base Case.** Let $\mathbf{c} = \langle q_1, \cdots, \square, \cdots, q_k\rangle \xrightarrow{a} p$ be a context. Then

$$\boldsymbol{\beta}_{\mathcal{A}_{\text{PUSH}}}\left(\mathbf{c}[q]\right) = w\left(\langle q_1, \cdots, q, \cdots, q_k\rangle \xrightarrow{a} p\right) \otimes \rho_{\text{PUSH}}\left(p\right) \tag{1.88}$$

and

$$\boldsymbol{\beta}_{\mathcal{A}_{\text{PUSH}}}\left(\mathbf{c}[q']\right) = w\left(\langle q_1, \cdots, q', \cdots, q_k\rangle \xrightarrow{a} p'\right) \otimes \rho_{\text{PUSH}}\left(p'\right). \tag{1.89}$$

As we proved in Lemma 1.11.2, the transitions $\langle q_1, \cdots, q, \cdots, q_k\rangle \xrightarrow{a/w} p$ and $\langle q_1, \cdots, q', \cdots, q_k\rangle \xrightarrow{a/w} p'$ must be weighted equally in $\mathcal{A}_{\text{PUSH}}$ since $q \sim_\delta q'$ and every other state in the left-hand side is equivalent to itself. Additionally, the final weights of p and p' are equal to $\mathbf{1}$ in $\mathcal{A}_{\text{PUSH}}$. This implies that $\boldsymbol{\beta}_{\mathcal{A}_{\text{PUSH}}}\left(\mathbf{c}[q]\right) = \boldsymbol{\beta}_{\mathcal{A}_{\text{PUSH}}}\left(\mathbf{c}[q']\right)$.

Inductive Step. Let $\mathbf{c}$ be a context such that $\boldsymbol{\beta}_{\mathcal{A}_{\text{PUSH}}}\left(\mathbf{c}[q]\right) = \boldsymbol{\beta}_{\mathcal{A}_{\text{PUSH}}}\left(\mathbf{c}[q']\right)$ for any equivalent $q, q' \in Q$. Additionally, let $\mathbf{c}' = a\left\langle q_1, \cdots, \square, \cdots, q_k\right\rangle$ be another context that has height 1. We will prove that $\boldsymbol{\beta}_{\mathcal{A}_{\text{PUSH}}}\left(\mathbf{c}[\mathbf{c}'][q]\right) = \boldsymbol{\beta}_{\mathcal{A}_{\text{PUSH}}}\left(\mathbf{c}[\mathbf{c}'][q']\right)$. By $\mathbf{c}[\mathbf{c}']$ we mean the context obtained by substituting $\mathbf{c}'$ into $\mathbf{c}$. Since the automaton is deterministic, there exists a single transition labeled with symbol a for each of the left-hand sides $\langle q_1, \cdots, q, \cdots, q_k\rangle$ and $\langle q_1, \cdots, q', \cdots, q_k\rangle$. We assume that $\langle q_1, \cdots, q, \cdots, q_k\rangle \xrightarrow{a/w} p$ and $\langle q_1, \cdots, q', \cdots, q_k\rangle \xrightarrow{a/w} p$ are these transitions. Notice that p and p' are also be equivalent if q and q' are equivalent, thus $\boldsymbol{\beta}_{\mathcal{A}_{\text{PUSH}}}\left(\mathbf{c}[p]\right) = \boldsymbol{\beta}_{\mathcal{A}_{\text{PUSH}}}\left(\mathbf{c}[p']\right)$ by the

inductive hypothesis. Putting these together, we get the following:

$$\boldsymbol{\beta}_{\mathcal{A}_{\text{PUSH}}}\left(\mathbf{c}[\mathbf{c}'][q]\right) = w\left(\langle q_1, \cdots, q, \cdots, q_k\rangle \xrightarrow{a/w} p\right) \otimes \boldsymbol{\beta}_{\mathcal{A}_{\text{PUSH}}}\left(\mathbf{c}[p]\right) \tag{1.90}$$

$$= w\left(\langle q_1, \cdots, q', \cdots, q_k\rangle \xrightarrow{a/w} p'\right) \otimes \boldsymbol{\beta}_{\mathcal{A}_{\text{PUSH}}}\left(\mathbf{c}[p']\right) \tag{1.91}$$

$$= \boldsymbol{\beta}_{\mathcal{A}_{\text{PUSH}}}\left(\mathbf{c}[\mathbf{c}'][q']\right). \tag{1.92}$$

This concludes the proof. ■

This implies that weighted minimization ca be done by following the same three steps we used for minimizing weighted finite-state automata.

Theorem 1.11.2. *Let $\mathcal{A}$ be a WFSTA. An equivalent minimal automaton $\mathcal{A}_{\text{MIN}}$ be computed using the following three steps:*

(i) Computing the coarsest partition $\mathcal{P}$ of the states on the unweighted version of $\mathcal{A}$;

(ii) Weight pushing using the WFSTA Quasi-Backwards algorithm to compute the pushing weights;

(iii) Unweighted minimization by refining the partition $\mathcal{P}$ on the automaton $\widetilde{\mathcal{A}_{\text{PUSH}}}$ in which the symbols and weights are treated as a single label.

Complexity The two partition refinement steps have a complexity of $\mathcal{O}\left(k|\Sigma||Q|\log(|Q|)\right)$, where k is the maximum arity of the symbols in Σ. Computing the pushing weights has a runtime of $\mathcal{O}\left(|\delta| + |Q|\right)$. Additionally, constructing the machines $\widetilde{\mathcal{A}_{\text{PUSH}}}$ and $\mathcal{A}_{\text{MIN}}$ can be done in linear time in the number of transitions, $\mathcal{O}\left(|\delta|\right)$. In total we get a complexity of $\mathcal{O}\left(k|\Sigma||Q|\log(|Q|)\right)$.

Bibliography

Hubert Comon, Max Dauchet, Rémi Gilleron, Florent Jacquemard, Denis Lugiez, Christof Löding, Sophie Tison, and Marc Tommasi. 2008. *Tree Automata Techniques and Applications*.

Thomas Hanneforth, Andreas Maletti, and Daniel Quernheim. 2017. Pushing for weighted tree automata. *CoRR*, abs/1702.00304.

Andreas Maletti. 2008. Myhill-nerode theorem for recognizable tree series revisited. In *Proceedings of the 8th Latin American Conference on Theoretical Informatics*, LATIN'08, page 106–120, Berlin, Heidelberg. Springer-Verlag.

Jonathan May and Kevin Knight. 2006. A better n-best list: Practical determinization of weighted finite tree automata. In *Proceedings of the Human Language Technology Conference of the NAACL, Main Conference*, pages 351–358, New York City, USA. Association for Computational Linguistics.