# Tree Automata

Willem van Summeren

April 8, 2018

# Contents

# 1 Introduction

Tree automata are automata that process trees rather than strings. A lot of research has been done on tree automata since they were conceptualized. Today almost all information concerning these automata is combined in one document: Tree Automata Techniques and Applications by H. Comon et al [1] .

Tree automata were introduced by Doner [2][3] and Thatcher and Wright[5][6] for proving the decidability of the weak second order theory of multiple successors, with their original definitions based on the algebraic approach. The style in which we represent tree automata has been introduced by J.E. Hopcroft and J.D. Ullman[4].

Whilst anything that can be done by tree automata should be possible to do using push down automata, there are things that tree automata can do more efficiently and in an easier to follow fashion. I.e: for checking if a tree is in the language $\{f^n(a)|n \in \mathbb{N}\}$ we only need two 2 rules and one state using tree automata, while it's PDA equivalent would be far more complex, since we will have to keep track of brackets.

Even though TATA [1] is considered to be the source material when researching tree automata, it is not always equally complete. The authors quickly glance over several subjects without fully explaining them, several theorems lack proof, some algorithms could have been added and a lot of subjects do not have any examples.

This document aims to add those missing proofs and algorithms to the first chapter of TATA [1], and clarify it's contents using clear examples. Concepts, definitions, algorithms will be used that are also described in TATA [1], yet this document can also be used as a standalone document for info on tree automata.

In this document we will start by defining some basics about trees, then define tree automata themselves. We will explain the difference between nondeterministic and deterministic tree automata, bottom-up and Top down tree automata, tree language properties and several algorithms to rewrite tree automata. Definitions and algorithms will be clarified using examples and proofs are added for the theorems we describe.

# 2 Preliminaries

In order to describe the functioning of tree automata, two sets will be defined here that will be used throughout this document. Most of these definitions originate from the preliminaries in TATA [1], pages 15 - 17.

## 2.1 Basic descriptions

**Definition 2.1.** *$\mathbb{N}$ is the set of all natural numbers, $\mathbb{N} = \{0, 1, 2, \ldots\}$., $n \in \mathbb{N}$ can be any non-negative integer.*

$\mathbb{N}$ is the set containing all natural numbers, $n$ is commonly used throughout this document as a variable in $\mathbb{N}$ unless otherwise defined at given time.

**Definition 2.2.** *$N^*$ is the set of finite strings over $\mathbb{N}$, $N^* = \{\varepsilon, 0, 1, \ldots .00, \ldots\}$, where an empty string is denoted by $\varepsilon$.*

$N^*$ is the set containing all possible concatenations of natural numbers.

## 2.2 Tree Representation

To describe trees, two definitions are introduced which are related to each other: the function $Arity(f)$ and the set $\mathcal{F}$.

**Definition 2.3.** *$\mathcal{F}$ is a finite set of symbols, $Arity(f)$ is a function mapping $\mathcal{F}$ to $\mathbb{N}$, for $f \in \mathcal{F}$, for which Arity(f) says how many arguments a symbol $f$ has.*

*$\mathcal{F}_n$ is the set of symbols $f$ for $arity(f) = n$. Symbols of $arity(f) = 0, 1, 2, 3, \ldots, n$ are called constant, unary, binary, ternary and n-nary respectively.*

*$\mathcal{F}$ should contain at least 1 constant.*

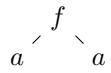
*We will shorten the descriptions of symbols by using only commas and brackets. A constant will be represented henceforth by using only the symbol of the constant itself: $f$. Symbols that are of higher arity will be represented using brackets: $f()$, and commas will be added to split arguments if the arity is above one: $f(,)$ means that $f$ is binary, $f(,,)$ is ternary and so on.*

Using these definitions we are able to describe trees. The requirement for $\mathcal{F}$ containing at least one constant is easily explained by the requirement for trees to have leaves: On the bottom of every branch in the tree there should be leaves, and only constants can act as leaves.

Trees can be either represented graphically, or as a formula, as we will see in the next examples.

---

**Example 2.4.** *Consider the tree $f(a, a)$. Its graphical representation is:*



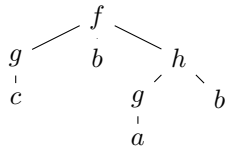

*For this tree $T$ we have $\mathcal{F} = \{f(,), a\}$.*

*$arity(f(,)) = 2$ and $arity(a) = 0$, therefore $\mathcal{F}_0 = \{a\}$, $\mathcal{F}_2 = \{f(,)\}$.*

*Put differently, $a$ is a constant and $f(,)$ is binary.*

---

**Example 2.5.** *Consider the tree $f(g(c), b, h(g(a), b))$. Its graphical representation is:*




*For this tree $T$ we have $\mathcal{F} = \{f(,,), g(), h(,), a, b, c\}$.*

*$arity(f(,,)) = 3$, $arity(g()) = 1$, $arity(h(,)) = 2$, $arity(a) = 0$, $arity(b) = 0$ and $arity(c) = 0$.*

*Thus $\mathcal{F}_0 = \{a, b, c\}$, $\mathcal{F}_1 = \{g()\}$, $\mathcal{F}_2 = \{h(,)\}$, $\mathcal{F}_3 = \{f(,,)\}$.*

*Put differently; $a, b$ and $c$ are constants, $g()$ is unary, $h(,)$ is binary and $f(,,)$ is ternary.*

---

The next example will show that a tree cannot be completed without constants.

---

**Example 2.6.** *Given a set $\mathcal{F} = \{f()\}$*
*The (bottom of a) tree made using this set would look like:*


*This tree would be incomplete, since $f$ still requires an argument. Given this set the only option would be giving it $f$ as argument resulting in the tree:*

*Hence repeating the problem. The only kind of symbol that does not require arguments is a constant, therefore constants are required to complete a tree.*

---

When looking at the n-nary symbols in previous examples it should be noted that these functions have several different options as argument, the arguments are variable. $f(,)$ is in fact $f(x, y)$ when not being shortened. Therefore a definition for variables is required.

**Definition 2.7.** *$V$ is the set of variables. $V$ and $\mathcal{F}_0$ should be disjoint. In this document, $x, y$ and $z$ are elements from $V$ unless stated otherwise.*

For means of clarity, $\mathcal{F}_0$ and $V$ should be disjoint, otherwise it would be unclear whether the character that is in both sets is either the constant or the variable, much like when using the character 1 as a variable regular math.

**Definition 2.8.** *The set $T(\mathcal{F}, V)$ of terms over $\mathcal{F}$ and $V$ is the smallest set defined by:*

- $\mathcal{F}_0 \subseteq T$
- $V \subseteq T$
- *if $p \geq 1, f \in \mathcal{F}_p$ and $t_1, \ldots, t_p \in T(\mathcal{F}, V)$, then $f(t_1, \ldots, t_p) \in T(\mathcal{F}, V)$*

*If $V = \emptyset$ then $T(\mathcal{F}, V)$ can be written as $T(\mathcal{F})$, of which its terms are called ground terms.*

*If each variable occurs at most once in $t \in T(\mathcal{F}, V)$, then $t$ is **linear**.*

---

**Example 2.9.** *Consider the terms $h, f(x), g(x, x)$ and $g(x, y)$. Here $h$ is a ground term, and terms $h, f(x)$ and $g(x, y)$ are linear, while $g(x, x)$ is not.*

---

**Definition 2.10.** *A **(ground) substitution** $\sigma$ is a mapping from $V$ into $T(\mathcal{F}, V)$, where the amount of variables mapped to themselves is finite.*

*The subset of variables $x \in V$ such that $\sigma(x) \neq x$ is called the **domain** of substitution $\sigma$.*

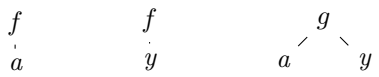
*For every $i$ between 1 and $n$, $\{x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n\}$ maps $x_i \in V$ on $t_i \in T(\mathcal{F}, V)$.*

We shall now clarify substitution with an example.

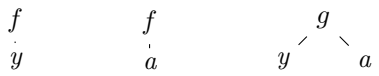
---

**Example 2.11.** *Given $\mathcal{F} = \{f(), g(,), a\}$ and $V = \{x, y\}$. Let us consider the terms $t_1 = f(x), t_2 = f(y)$ and $t_3 = g(x, y)$ and the substitutions $\sigma_1 = \{x \leftarrow a\}$ and $\sigma_2 = \{x \leftarrow y, y \leftarrow a\}$.*

*First we get the results of $t_1\sigma_1 = f(x)\{x \leftarrow a\}$ (left), $t_2\sigma_1 = f(y)\{x \leftarrow a\}$ (middle) and $t_3\sigma_1 = g(x, y)\{x \leftarrow a\}$(right):*



*Next are the results of $t_1\sigma_2 = f(x)\{x \leftarrow y, y \leftarrow a\}$ (left), $t_2\sigma_2 = f(y)\{x \leftarrow y, y \leftarrow a\}$ (middle) and $t_3\sigma_2 = g(x,y)\{x \leftarrow y, y \leftarrow a\}$(right):*



---

**Definition 2.12.** *A linear term $C \in T(\mathcal{F}, V_n)$, in which $V_n$ is a sequence of $x_1, \ldots, x_n$ , is called a **Context**.*

*The expression $C[t_1, \ldots, t_n]$ for $t_1, \ldots, t_n \in T(\mathcal{F})$ denotes the term in $T(\mathcal{F})$ obtained by applying $C[t_1, \ldots, t_n] = C\{x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n\}$, which means that for $1 \leq i \leq n$ every variable $x_i$ is replaced by $t_i$. A context is **trivial** if it is a variable.*

---

**Example 2.13.** *Given $\mathcal{F} = \{a, b, f(), g(,)\}$ and $V = \{x, y\}$, some examples of $C \in T(\mathcal{F}, V)$ are: $C[x], C[f(a)], C[f(b)], C[f(y)], C[g(a,a)]$ and $C[G(a,b)]$.*

---

**Definition 2.14.** *$K^n(\mathcal{F})$ denotes the set of contexts over n variables.$K(\mathcal{F})$ denotes the set of contexts containing a single variable, meaning $K^1(\mathcal{F}) = K(\mathcal{F})$.*

*Given $C \in K(\mathcal{F})$, $C^0$ is the trivial context, $C^1 = C$ and $C^n = C^{n-1}[C]$ for $n > 1$ is a context in $K(\mathcal{F})$.*

In order to describe properties of trees, we define several functions that will be used later on.

**Definition 2.15.** *$Pos(t) \subseteq N^*$ is a set of **positions** satisfying the properties:*

- *$Pos(t)$ is nonempty and prefix-closed.*
- *$\forall p \in Pos(t)\ \ t(p) \in \mathcal{F}_n, n \geq 1 \implies \{j | pj \in Pos(t)\} = \{1, \ldots, n\}$*
- *$\forall p \in Pos(t)\ \ t(p) \in V \cup \mathcal{F}_0 \implies \{j | pj \in Pos(t)\} = \emptyset$*

*A position that complies to $\forall j \in \mathbb{N}, pj \notin Pos(t)$ is called a **frontier position**, the set of which is denoted as $FPos(t)$. Positions in t such that $t(p) \in V$ are called a **variable position**, denoted as $VPos(t)$. $Head(t)$ is the root symbol of $t$.*

**Definition 2.16.** *The size of a term t is denoted by $||t||$, is inductively defined as:*

- *$||t|| = 0$ if $t \in V$*
- *$||t|| = 1$ if $t \in \mathcal{F}_0$*

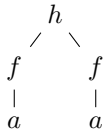
- $||t|| = 1 + \sum_{i \in \{1,\ldots,n\}} ||t_i||$ *if* $Head(t) \in F_n$

**Definition 2.17.** *The height of a term* $t$ *is denoted by* $Height(t)$*, is inductively defined as:*

- $Height(t) = 0$ *if* $t \in V$
- $Height(t) = 1$ *if* $t \in \mathcal{F}_0$
- $Height(t) = 1 + max(Height(t_i) | i \in \{1, \ldots, n\})$ *if* $Head(t) \in F_n$

---

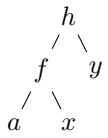
**Example 2.18.** *Given the tree:*




*In this tree the head is* $h$*, the set of frontier positions is* $\{11, 21\}$ *which are the positions of both* $a$*'s in the tree. The height of the tree is 3 and its size is 5.*

---

**Example 2.19.** *Given the tree:*




*In this tree the head is* $h$*, the set of frontier positions is* $\{11, 12, 2\}$*, and the set of variable positions is* $\{12, 2\}$*. The height of the tree is 3 and its size is 3.*

---

## 2.3 Trees without brackets

It is common practice to write down trees using brackets in order to keep them readable: when we read things $f(a, a)$ or $f(g(a), g(f(a, a)))$ we know exactly what that tree should look like.

It is also possible to write down trees without using brackets and commas, thus writing down the previous trees as $faa$ and $fgagfaa$.

In both representations, the leftmost branch of each tree is described first, and the symbols are described depth-first. In the representation using brackets this is enforced by use of brackets and commas, in the string representation this is only done by the order of the symbols. Since there are no brackets or commas to enforce the or represent the arity of used symbols, the arity of all symbols must be known in order to construct the represented tree.

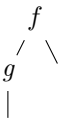
This property ensures that any given tree represented as string can only be interpreted one way, since the leftmost open branch must always be completed first.

---

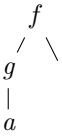
**Example 2.20.** *Given $arity(f) = 2, arity(g) = 1$ and $arity(a) = 0$, the tree of the string $fgagfaa$'s is constructed as follows:*

*First the $f$ is processed leaving $gagfaa$ as remainder of the string and building the tree:*

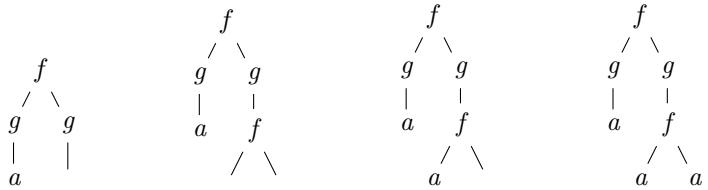

*Next we process the symbol $g$, which has arity 1, since it has to go to the leftmost open spot we get:*




*Next we must place the $a$, leaving $gfaa$ as the remainder string. Once again we must place it on the leftmost position:*



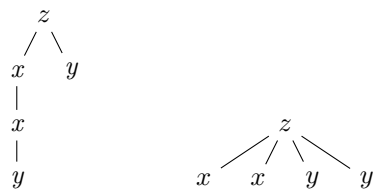

*When we continue the processing the string in using the same method, we get the products below:*



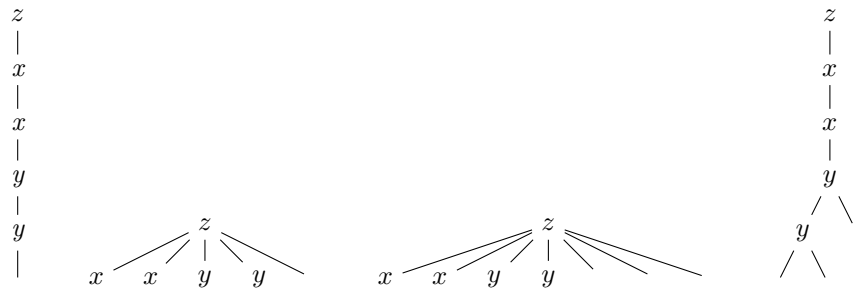

*The result is the same tree as represented by $f(g(a), g(f(a, a)))$.*

---

**Example 2.21.** *Given a tree string $zxxyy$ with no known arities, possible representations are:*




*A finite number of incomplete trees can also be made, such as:*




*Knowing our string is a valid and complete tree limits the number of possible trees, yet without knowing the arities of the symbols even the simple example above has 2 completely different possibilities. A string such as $fgfhaaba$ would have many more possible forms, and without further knowing the arities of the used symbols it is impossible to say which of these is meant.*

**Definition 2.22.** *We will now use a tuple $(s, o)$ in which $s$ is the string representation of a tree, and $o$ is the number of open subtree slots that still need to be filled. $s$ consists of symbols $f_n$ in which $n$ is the arity of that symbol.*

*Given a string $s$, a string $s'$ in which the first character $f_n$ has been removed and the amount of open slots $o$, we can reduce the string as follows:*

$$(s, o) \to (s', o - 1 + n) \qquad \textbf{\textit{r1}}$$

*Starting with any given string $s$, we start out with a value for $o$ of 1 since there is currently one open branch, which is the root.*

*Now if the string is a valid tree, there should be a reduction path:*

$$(s, 1) \xrightarrow{*} (\epsilon, 0)$$

*In which $\epsilon$ is an empty string. We can then define a function:*

$$toTree :: String \to Tree \vee error$$

*which produces a tree from a string if that strings represents a valid tree, meaning* $(s,1) \xrightarrow{*} (\epsilon,0)$ *holds, or returns an error whenever the tree is invalid, thus being when* $(s,1) \xrightarrow{*} (\epsilon, n \neq 0)$ *or* $(s,1) \xrightarrow{*} (s' \neq \epsilon, 0)$.

**Theorem 2.23.** *If the string s denotes a valid tree, then the procedure outlined in Definition 2.22 above yields:*

$$(s,1) \xrightarrow{*} (\epsilon,0)$$

---

**Proof of (2.23).** *Induction basis: If s consists of a single constant* $f_0$*, we get a starting scenario of* $(f_0,1)$*. If we apply r1 to this tuple we get:*

$$(f_n,1) \rightarrow (\epsilon,0)$$

*Induction Step: For* $s = f_k s'$,$k \geq 1$ *we get starting scenario* $(f_k s',1)$*. As induction hypothesis we take:* $(s',k) \xrightarrow{*} (\epsilon,0)$*. Applying r1 we get:*

$$(f_k s',1) \rightarrow (s',k)$$

*When we apply the Induction hypothesis we get:*

$$(f_k s',1) \rightarrow (s',k) \xrightarrow{*} (\epsilon,0)$$

*Thus completing the proof.*

---

# 3 Finite Tree Automata

## 3.1 General properties

### 3.1.1 Description

Section 1.1 in TATA describes several forms of tree automata: The Nondeterministic Finite Tree Automaton (NFTA), the Deterministic Finite tree Automaton, a NFTA with epsilon rules and Top down Tree Automaton (TDTA). We will start by defining NFTA's, since all rules for a NFTA also apply to DFTA's, yet the DFTA has some extra restrictions which NFTA's do not have.

We will first introduce a few definitions that apply to all tree automata.

**Definition 3.1.** *A tree automaton over $\mathcal{F}$ is a tuple $\mathcal{A} = (Q, \mathcal{F}, Q_f, \Delta)$, where:*

$Q$ *is a final set of states.*

$\mathcal{F}$ *is the set of symbols, which each have an arity.*

$Q_f$ *is the set of final states where $Q_f \subseteq Q$.*

$\Delta$ *is the finite set of transition rules.*

**Definition 3.2.** *Transition rules are of the form:*

$$f(q_1(x_1), \ldots, q_n(x_n)) \to q(f(x_1, \ldots, x_n))$$

*Where $n \geq 0$, $f \in \mathcal{F}$ with $arity(f) = n$, $q, q_1, \ldots, q_n \in Q$ and $x_1, \ldots, x_n \in V$*

Tree automata do not have an initial state, instead they have initial (transition) rules, which are transition rules handling transitions at the leaves of the tree.

A simple transition rule will be used in the example below to further the explanation of transition rules.

---

**Example 3.3.** *Consider the transition rule:*

$$a \to q_1(a)$$

*in which $a$ is a constant and $q_1$ is the resulting state of that transition, which has $a$ as an argument. Transitions may require a set underlying states to allow the transition.*

*Applying this rule on a tree consisting of only $a$, will result in the following tree:*

$$\begin{array}{c} q_1 \\ | \\ a \end{array}$$

---

**Definition 3.4.** $\underset{\mathcal{A}}{\longrightarrow}$ *means a transition is made on a tree automaton using a transition rule from* $\Delta$.

*Let* $\mathcal{A} = (Q, \mathcal{F}, Q_f, \Delta)$ *be a Tree Automaton over* $\mathcal{F}$*, The relation* $t \underset{\mathcal{A}}{\longrightarrow} t'$ *for trees* $t$ *and* $t'$ *is defined as follows:*

*Let* $t, t' \in T(\mathcal{F} \cup Q)$

$$t \underset{\mathcal{A}}{\longrightarrow} t' := \begin{cases} \exists C \in K(\mathcal{F} \cup Q), \exists u_1, \ldots, u_n \in T(\mathcal{F})), \\ \exists f(q_1(x_1), \ldots, q_n(x_n)) \to q(f(x_1, \ldots, x_n)) \in \Delta, \\ t = C[f(q_1(u_1), \ldots, q_n(u_n))], \\ t' = C[q(f(u_1, \ldots, u_n))]. \end{cases} \tag{1}$$

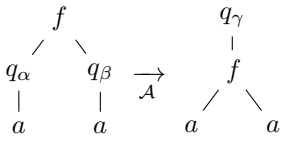
**Definition 3.5.** $\overset{*}{\underset{\mathcal{A}}{\longrightarrow}}$ *is the transitive and reflexive closure of* $\underset{\mathcal{A}}{\longrightarrow}$

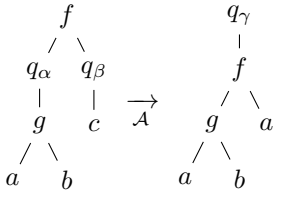
---

**Example 3.6.** *We consider the tree automaton below with* $\mathcal{F} = \{f(,), g(,), a, b, c\}$ *and* $f(q_\alpha(x), q_\beta(y)) \to q_\gamma(f(x, y)) \in \Delta$*. Applying the transition rule:*

$$f(q_\alpha(x), q_\beta(y)) \to q_\gamma(f(x, y))$$

*on the trees below we get:*



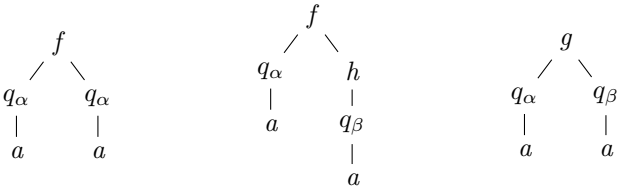




---

This transition is not valid for trees that do no meet the requirements, thus those that contain symbols or parts of trees not described in the given transition rules.

---

**Example 3.7.** *We consider the tree automaton below with* $\mathcal{F} = \{f(,), g(,), h(), a, b, c\}$ *and* $f(q_\alpha(x), q_\beta(y)) \to q_\gamma(f(x, y)) \in \Delta$*. Applying the transition rule* $f(q_\alpha(x), q_\beta(y)) \to q_\gamma(f(x, y))$ *on the trees below is not possible:*




---

### 3.1.2 Acceptance

**Definition 3.8.** *A tree is accepted by a tree automaton if there is a computation that halts in a final state, where said state is above the tree, or in formula:*

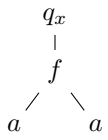
$$t \xrightarrow[\mathcal{A}]{*} q_f(t)$$

*Where* $q_f \in Q_f$.

Tree automata do not accept a tree when computations halt before the end of the tree has been reached, or the automaton produces a result $q_x(t)$ in which $q_x$ is not a final state.

---

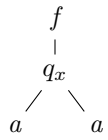
**Example 3.9.** *The following automaton with* $\mathcal{F} = \{f(,), a\}$ *that has come to a halt:*




*In this tree automaton calculations have halted with the state* $q_x$ *on top of the tree. In the case* $q_x$ *is a final state (*$q_x \in Q_f$*) the input tree is accepted. In the case that* $q_x$ *is not a final state (*$q_x \notin Q_f$*) the input tree is rejected.*

---

**Example 3.10.** *Looking at the following automaton with* $\mathcal{F} = \{f(,), a\}$ *that has come to a halt:*




*In this example it does not matter whether* $q_x$ *is a final state since the full tree has not been processed, or in other words,* $q_x$ *is not the top of the tree. Therefore it does not satisfy the acceptance rule* $t \xrightarrow[\mathcal{A}]{*} q_f(t)$.

---

**Definition 3.11.** *The tree language* ***recognized*** *by* $\mathcal{A}$*, which is denoted as* $\mathcal{L}(\mathcal{A})$*, is the set of all ground terms accepted by* $\mathcal{A}$*. A set* $L$ *of ground terms is* ***recognizable*** *if for some* $A$*:*

$$L = \mathcal{L}(\mathcal{A})$$

*Two tree automata are* ***equivalent*** *if they recognize the same tree language.*

### 3.1.3 Tree Automata as a Ground Rewrite System

In the previous sections transition rules have been defined, in such a way that the original tree is preserved when processed by a Tree Automata, only adding states and moving them up in the tree.

TATA (section 1.1, p21) also explains how to write down trees as a ground rewrite system, and continues using this method throughout the document.
We can also define these transitions as a ground rewrite system, in which the original tree is not preserved but rewritten by states.

**Definition 3.12.** *In a tree automaton that is written as a ground rewrite system, $\Delta$ contains only rules of the form:*

$$f(q_1 \dots, q_n) \to q$$

*These rules are named **ground transition rules**. A tree automaton using ground state transition rules accepts a term $t$ if:*

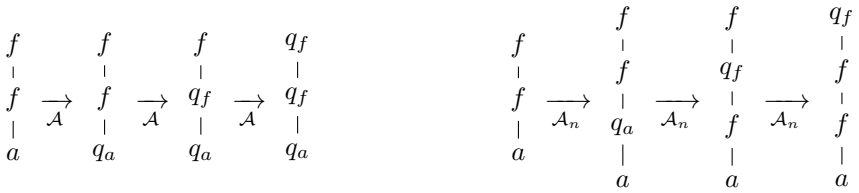
$$t \xrightarrow[\mathcal{A}]{*} q_f \text{ with } q_f \in Q_f$$

The language acceptance of Tree Automata is unaffected by whether ground transition rules or normal transition rules are used, since the automata will still work upward and only has rewrite rules for terms, but not for states. For this reason, no extra transitions are made in an automaton using ground transition rules.

The resulting tree however will look different, it will no longer be the original tree with an amount of states added to it, rather a tree with states as far as the tree could be processed.

---

**Example 3.13.** *We consider the tree automata $\mathcal{A}$ (left) and $\mathcal{A}_n$ (right) below, both with $\mathcal{F} = \{f(), a\}$, $q_a, q_f \in Q$ and $q_f \in Q_f$. For $\mathcal{A}$ we have $a \to q_a, f(q_a) \to q_f \in \Delta$ and for $\mathcal{A}_n$ we have $a \to q_a(a), f(q_a(x)) \to q_f(f(x)) \in \Delta_n$ which are the same rules in both forms.*



*In either case the input tree has been accepted, yet the resulting trees are different.*

---

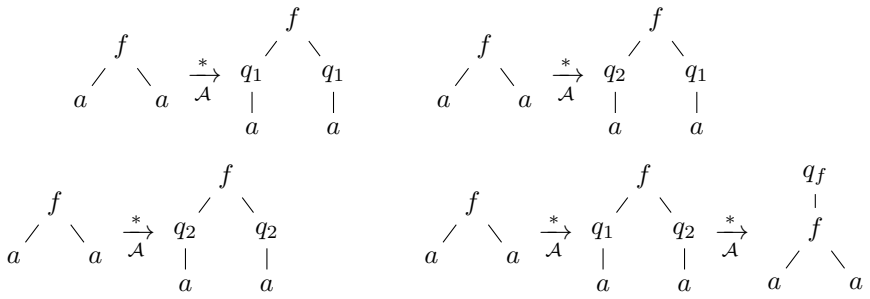
## 3.2 Nondeterministic Finite Tree Automata

A tree automaton that only follows the rules listed in the previous subsection is called a Nondeterministic Finite Tree Automaton. A NFTA is a tree automaton that accepts trees if any sequence of transition rules in $\Delta$ leads to satisfying the acceptance rule.

For any (processed) input multiple transitions could occur, and multiple transitions can lead to the same state. A Nondeterministic Finite Tree Automaton can be compared to a nondeterministic finite automaton. Given an (processed) input, multiple transitions can occur. If any of these satisfy $t \xrightarrow[\mathcal{A}]{*} q_f(t)$, then the input tree is accepted.

---

**Example 3.14.** *Given an NFTA $\mathcal{A}$ with $\mathcal{F} = \{f(,)a\}, \Delta = \{a \to q_1(a), a \to q_2(a), f(q_1(x), q_2(y)) \to q_f(f(x,y))\}$, where $q_f \in Q_f$, and an input tree $t = f(a,a)$. In this example the following transitions can occur:*




*Since one of these computations lead to an accepting state above the input tree, $t$ is accepted.*

---

Another variant of a NFTA exists, which has rules to alter a state without processing any symbols of a tree. This kind of NFTA, named NFTA-$\varepsilon$, will be described in a later section.

## 3.3 Deterministic Finite Tree Automata

Section 1.1 of TATA[1] also introduces the notion of a Deterministic Finite Tree Automaton. The DFTA follows the same rules as the NFTA, but has some extra restrictions.

**Definition 3.15.** *A **DFTA** over $\mathcal{F}$ is a tuple $\mathcal{A} = (Q, \mathcal{F}, Q_f, \Delta)$ in which there is at most one transition rule for every read input.*

*A DFTA is unambiguous, which means that for every transition rule $l \to r \in \Delta$ there are no 2 rules with the same term on the left side of the rule. In other words: There is only one single output for every input tree in a DFTA.*

---

**Example 3.16.** *Given three tree automata all with:*

$$Q = \{q_1, q_2\}$$

$$\mathcal{F} = \{f(), a, b\}$$

$$Q_f = \{q_2\}$$

*Each of these automaton $A_i$ has its own transition rule set $\Delta_i$.*
*The sets of transition rules are the following:*

$$\Delta_1 = \{a \to q_1(a), \quad b \to q_2(b), \quad f(q_1(x)) \to q_2(f(x)), \quad f(q_2(x)) \to q_2(f(x))\}$$

$$\Delta_2 = \{a \to q_1(a), \quad b \to q_1(b), \quad f(q_1(x)) \to q_2(f(x)), \quad f(q_2(x)) \to q_1(f(x))\}$$

$$\Delta_3 = \{a \to q_1(a), \quad b \to q_1(b), \quad f(q_1(x)) \to q_1(f(x))\}$$

*$\mathcal{A}_1, \mathcal{A}_2$ and $\mathcal{A}_3$ are all DFTA's since they comply to the rules set in Subsection 3.1.1. and definition 3.15. $\mathcal{A}_3$ will not be able to accept any trees, yet that is not a requirement to either a tree automaton or a DFTA.*

---

**Example 3.17.** *Given tree automata $\mathcal{A}_4$ and $\mathcal{A}_5$ with:*

$$Q = \{q_1, q_2, q_f\}$$

$$\mathcal{F} = \{f(), a, b\}$$

$$Q_f = \{q_f\}$$

*The sets of transition rules are the following:*

$$\Delta_4 = \{a \to q_1(a), \quad a \to q_2(a), \quad f(q_1(x)) \to q_3(f(x)), \quad f(q_2(x)) \to q_f(f(x))\}$$

$$\Delta_5 = \{a \to q_1(a), \quad b \to q_1(b), \quad f(q_1(x)) \to q_2(f(x)), \quad f(q_1(x)) \to q_f(f(x))\}$$

*Both $\mathcal{A}_4$ and $\mathcal{A}_5$ are not DFTA's. In $\mathcal{A}_4$ we can see 2 transition rules for a read character $a$, and therefore it is ambiguous, and in $\mathcal{A}_5$ we have the same issue for $f(q_1(x))$. They are still valid NFTA's however.*

---

**Definition 3.18.** *A **complete DFTA** is a DFTA which has at least one rule $f(q_1(x_1), \ldots, q_n(x_n)) \to q(f(x_1, \ldots, x_n)) \in \Delta$ for all $n \geq 0, f \in \mathcal{F}$, and $q_1, \ldots, q_n \in Q$.*

---

**Example 3.19.** *Looking back at the automata in examples 3.16 and 3.17, we can now state that $\mathcal{A}_1$ and $\mathcal{A}_2$ are complete DFTA's, $\mathcal{A}_3$, $\mathcal{A}_4$ and $\mathcal{A}_5$ are not, this is because:*

*In* $\mathcal{A}_1$ *there is a transition for every possible input.*

*In* $\mathcal{A}_2$ *there is a transition for every possible input.*

*In* $\mathcal{A}_3$ *there is no transition for* $f(q_2(x))$

*In* $\mathcal{A}_4$ *is not a DFTA.*

*In* $\mathcal{A}_5$ *is not a DFTA.*

---

In the previous subsection we stated that the NFTA can be seen as the NFA for trees. Likewise, a complete deterministic tree automaton (DFTA) can be seen as the tree equivalent of the Deterministic Finite Automaton. Like in a DFA, given a read piece of the input (or automaton processed input in the case of tree automata) there is exactly one possible transition.

Any DFTA can be transformed into a complete DFTA accepting the same language, which TATA[1] has made no mention of. The idea is simple: Add a dead state (a state that states the machine is stuck) to the machine and add transition rules for any read tree for which there are none, with each of these rules going to the dead state.

**Algorithm 3.20.** *Completion Algorithm*
***COMPLETE*** *(DFTA* $\mathcal{A} = (Q_d, \mathcal{F}, Q_{df}, \Delta_d)$*)*
$Q_c = Q_d \cup Q_{fail}$ *where* $Q_{fail} \notin Q_d, Q_{fail} \notin Q_{df}$
$\Delta_c = \Delta_d$
***While*** *(rules can be added to* $\Delta_c$*)*
{
$\Delta_c = \Delta_c \cup \{f(q_1(x_1), \ldots, q_n(x_n)) \to q_{fail}(f(x_1, \ldots, x_n))\}$
***where*** $f \in \mathcal{F}, \quad q_1, \ldots, q_n \in Q_c$
$\nexists_{q \in Q_c}(f(q_1(x_1), \ldots, q_n(x_n)) \to q(f(x_1, \ldots x_n)) \in \Delta_c)$
}
***return*** *DFTA* $\mathcal{A}_c = (Q_c, \mathcal{F}, Q_{df}, \Delta_c)$

---

**Example 3.21.** *Given a DFTA with:*

$\mathcal{F} = \{f(,), g(), a\}$.

$Q_d = \{q_a, q_f, q_g\}$.

$Q_{df} = \{q_g\}$.

$\Delta_d = \{a \to q_a(a), \quad f(q_a(x), q_a(y)) \to q_f(f(x, y)) \quad g(q_f(x)) \to q_g(g(x))\}$.

*When we start the completion algorithm3.20, we first get sets* $Q_c = Q_d \cup q_{fail}$ *and* $\Delta_c = \Delta_d$*. Then we add rules to* $\Delta_c$ *for trees that do not appear on any left hand side of any transition rules, all leading to the state* $q_{fail}$*:*

$$g(q_a(x)) \to q_{fail}(g(x)) \qquad g(q_f(x)) \to q_{fail}(g(x))$$
$$g(q_g(x)) \to q_{fail}(g(x)) \qquad g(q_{fail}(x)) \to q_{fail}(g(x))$$
$$f(q_a(x), q_f(y)) \to q_{fail}(f(x,y)) \qquad f(q_f(x), q_a(y)) \to q_{fail}(f(x,y))$$
$$f(q_a(x), q_g(y)) \to q_{fail}(f(x,y)) \qquad f(q_g(x), q_a(y)) \to q_{fail}(f(x,y))$$
$$f(q_a(x), q_{fail}(y)) \to q_{fail}(f(x,y)) \qquad f(q_{fail}(x), q_a(y)) \to q_{fail}(f(x,y))$$
$$f(q_f(x), q_f(y)) \to q_{fail}(f(x,y)) \qquad f(q_g(x), q_g(y)) \to q_{fail}(f(x,y))$$
$$f(q_f(x), q_g(y)) \to q_{fail}(f(x,y)) \qquad f(q_g(x), q_f(y)) \to q_{fail}(f(x,y))$$
$$f(q_f(x), q_{fail}(y)) \to q_{fail}(f(x,y)) \qquad f(q_{fail}(x), q_f(y)) \to q_{fail}(f(x,y))$$
$$f(q_g(x), q_{fail}(y)) \to q_{fail}(f(x,y)) \qquad f(q_{fail}(x), q_g(y)) \to q_{fail}(f(x,y))$$
$$f(q_{fail}(x), q_{fail}(y)) \to q_{fail}(f(x,y))$$

*The result is a new complete deterministic finite tree automaton* $(Q_c, \mathcal{F}, Q_{df}, \Delta_c)$.

---

**Theorem 3.22.** *For every DFTA, there exists a complete DFTA that accepts the same language.*

---

**Proof of** (3.22)**.** *Using algorithm 3.20 we can construct a complete DFTA* $\mathcal{A}_c$ *using any input DFTA* $\mathcal{A}_d$*. In order for Theorem 3.22 to hold we must show that* $\mathcal{A}_c$ *accepts a tree* $t$ *only and only if it is accepted by* $\mathcal{A}_d$*. We can split this in 2 cases:*

> $\underline{t \in \mathcal{L}(\mathcal{A}_d)}$ : *Any tree* $t$ *accepted by* $\mathcal{A}_d$ *is also accepted by* $\mathcal{A}_c$*, since it uses the same transition rules and states for processing that tree, resulting in the same final tree. A rule going to state* $q_{fail}$ *will never be accessed processing this tree since the original automaton* $\mathcal{A}_d$ *will have had rules for any transactions in this tree. It is also impossible for a DFTA to infer transition rules on any fully processed tree.*
>
> $\underline{t \notin \mathcal{L}(\mathcal{A}_d)}$ : *Which can have one of two reasons:*
>
> > $\underline{t \xrightarrow[\mathcal{A}_d]{*} q(t), q \notin Q_{df}}$ : *In this case* $\mathcal{A}_c$ *will process* $t$ *exactly the same resulting in the same state and rejecting* $t$*.*
> >
> > $\underline{t \not\xrightarrow[\mathcal{A}_d]{*} q(t)}$ : *This case is the one where* $\mathcal{A}_c$ *differs from* $\mathcal{A}_d$*. Where* $\mathcal{A}_d$ *gets stuck,* $\mathcal{A}_c$ *transfers to a state* $q_{fail}$ *(which is a sink), and from that point onward keeps pushing state* $q_{fail}$ *upward. Since* $q_{fail} \notin Q_{df}$*, the end result* $q_{fail}(t)$ *will not be accepted.*

*Therefore* $\mathcal{A}_c$ *and* $\mathcal{A}_d$ *accept the same language.*

## 3.4 NFTA versus DFTA

Whilst the NFTA and DFTA are slightly different, for every NFTA there is a equivalent DFTA and vice versa. This would mean there is no difference in their expressive power, and that different variants only simplify or change the form or Tree Automaton, not change it's acceptance.

We can state that every DFTA is a valid NFTA since both are tuples $\mathcal{A} = (Q, \mathcal{F}, Q_f, \Delta)$, and a NFTA has even less restrictions than a DFTA. Therefore the most easily made NFTA from a DFTA is the DFTA itself without any changes. Smaller and simpler versions might be made in some instances, but this fact does show that for every DFTA there is a NFTA equivalent.

Any NFTA has an equivalent DFTA. A NFTA can be converted into a DFTA using the algorithm below. In this algorithm $Q_d$ is the newly formed set of state sets, which is a subset of the power set of $Q_n$, $Q_d \subseteq \mathcal{P}(Q_n)$ in short. This enabled the nondeterministic transitions to be converted, making them transition to a set of states containing all states the NFTA could transition to.

TATA (1.1, p26) proposes an algorithm to transform NFTA's into an equivalent DFTA. Below we have an altered version of this algorithm, which does not use tree automata as a ground rewrite system.

**Algorithm 3.23.** *Determinization Algorithm*
***DET*** *(NFTA* $\mathcal{A} = (Q_n, \mathcal{F}, Q_{nf}, \Delta_n)$*)*
$Q_d = \emptyset$
$\Delta_d = \emptyset$
***While*** *(rules can be added to* $\Delta_d$*)*
{
$Q_d = Q_d \cup \{s\}$
$\Delta_d = \Delta_d \cup \{f(s_1(x_1), \ldots, s_n(x_n)) \to s(f(x_1, \ldots, x_n))\}$
***where*** $f \in \mathcal{F}, \quad s_1, \ldots, s_n \in Q_d$
$s = \{q \in Q_n | \exists (q_1 \in s_1, \ldots, q_n \in s_n),$
$f(q_1(x_1), \ldots, q_n(x_n)) \to q(f(x_1, \ldots x_n)) \in \Delta_n\}$ ***and*** $s \neq \emptyset$
}
$Q_{df} = \{s \in Q_d | s \cap Q_f \neq \emptyset\}$
***return*** *DFTA* $\mathcal{A}_d = (Q_d, \mathcal{F}, Q_{df}, \Delta_d)$

Using this algorithm NFTA's can be converted into equivalent DFTA's. The resulting tree automaton is a DFTA, since it has no nondeterministic steps. The algorithms turns nondeterministic steps into a single transition towards the set of all resulting states of the nondeterministic transitions. The resulting DFTA is not complete, since there will be sets of states which do not occur on either side of the transition rules.

---

**Example 3.24.** *We use the determinization algorithm on the NFTA in Example 3.14. First we add a subscript n to the set of states and the set of transition rules to more easily identify them as belonging to the NFTA,* $Q_n = \{q_1, q_2, q_f\}, \Delta_n = \{a \to q_1(a), a \to q_2(a), f(q_1(x), q_2(y)) \to q_f(f(x, y))\}$.

*We first set* $Q_d$ *and* $\Delta_d$*, the set of states and the set of transition rules for the DFTA, to be empty.*

*We then look for possible s. Since* $Q_n$ *is empty only rules adding constants can be added, for which we find the rules* $a \to q_1(a)$ *and* $a \to q_2(a)$*, hence for this formula we take* $f = a, s = \{q_1, q_2\}$ *and then expand* $Q_d$ *with* $\{q_1, q_2\}$*, and* $\Delta_d$ *with* $a \to \{q_1, q_2\}(a)$.

*This results in the sets* $\Delta_d = \{a \to \{q_1, q_2\}(a)\}$ *and* $Q_d = \{q_1, q_2\}$*. Now there is one last rule we can add. if we pick* $f = f(,), s = q_f, s_1 = q_1$ *and* $s_2 = q_2$ *we can add the rule* $f(\{q_1, q_2\}(x), \{q_1, q_2\}(y)) \to \{q_f\}(f(x, y))$ *to* $\Delta_d$*, and* $\{q_f\}$ *to* $Q_d$.

*This will result in* $\Delta_d = \{a \to \{q_1, q_2\}(a), f(\{q_1, q_2\}(x), \{q_1, q_2\}(y)) \to \{q_f\}(f(x, y))\}$ *and* $Q_d = \{\{q_1, q_2\}.\{q_f\}\}$*. Finally* $Q_{df} = \{q_f\}$ *is set in the final line and DFTA* $\mathcal{A}_d = (Q_d, \mathcal{F}, Q_{df}, \Delta_d)$ *is completed.*

---

**Example 3.25.** *In this example we will only show what is produced each iteration in a table. For this example we will use the following data:*

$Q_n = \{q_1, \ldots q_5\}$
$Q_f = \{q_4, q_5\}$
$\mathcal{F} = \{a, b, c, f(), g(,)\}$
$\Delta_n = \{a \to q_1(a), b \to q_1(b), b \to q_2(b), c \to q_2(c), f(q_1(x)) \to q_3(f(x)),$
$f(q_2(x)) \to q_3(f(x)), g(q_1(x), q_1(y) \to q_4(g(x, y)), g(q_3(x), q_1(y)) \to q_5(g(x))\}$

*We will now display all data from using the algorithm in the table below, in which we show for each iteration what symbol is processed, the* $\{s\}$ *that is evaluated and the rule that is added.*

| | $\mathcal{F}$ | $\{s\}$ | *Rule* |
|---|---|---|---|
| *0* | | | |
| *1* | $a$ | $\{q_1\}$ | $a \to \{q_1\}(a)$ |
| *2* | $b$ | $\{q_1, q_2\}$ | $b \to \{q_1, q_2\}(b)$ |
| *3* | $c$ | $\{q_2\}$ | $c \to \{q_2\}(c)$ |
| *4* | $f()$ | $\{q_3\}$ | $f(\{q_1\}(x)) \to \{q_3\}(f(x))$ |
| *5* | $f()$ | $\{q_3\}$ | $f(\{q_1, q_2\}(x)) \to \{q_3\}(f(x))$ |
| *6* | $f()$ | $\{q_3\}$ | $f(\{q_2\}(x)) \to \{q_3(f(x))\}$ |
| *7* | $g(,)$ | $\{q_4\}$ | $g(\{q_1\}(x), \{q_1\}(y)) \to \{q_4\}(g(x, y))$ |
| *8* | $g(,)$ | $\{q_4\}$ | $g(\{q_1\}(x), \{q_1, q_2\}(y)) \to \{q_4\}(g(x, y))$ |
| *9* | $g(,)$ | $\{q_4\}$ | $g(\{q_1, q_2\}(x), \{q_1\}(y)) \to \{q_4\}(g(x, y))$ |
| *10* | $g(,)$ | $\{q_4\}$ | $g(\{q_1, q_2\}(x), \{q_1, q_2\}(y)) \to \{q_4\}(g(x, y))$ |
| *11* | $g(,)$ | $\{q_5\}$ | $g(\{q_3\}(x), \{q_1\}(y)) \to \{q_5\}(g(x, y))$ |
| *12* | $g(,)$ | $\{q_5\}$ | $g(\{q_3\}(x), \{q_1, q_2\}(y)) \to \{q_5\}(g(x, y))$ |

*$Q_d$ is then the set of all items listed in the $\{s\}$ column, $\Delta_d$ the set of all rules listed in the rule column and $Q_{df}$ is the set of all states $Q_d$ that contain states in $Q_f$, thus $Q_{df} = \{\{q_4\}, \{q_5\}\}$.*

---

**Theorem 3.26.** *Using Algorithm 3.23, any NFTA $\mathcal{A}_n$ can be converted in an equivalent DFTA $\mathcal{A}_d$, which accepts the same language ($\mathcal{L}(\mathcal{A}_n) = \mathcal{L}(\mathcal{A}_d)$).*

*Given a term $t$, if a transition to a state $q$ can be made in the NFTA, a transition is made in the DFTA towards a set containing $q$ and vice versa:*

$$\forall q \in Q_n (t \xrightarrow[\mathcal{A}_n]{*} q(t)) \iff (t \xrightarrow[\mathcal{A}_d]{*} s(t)) \textbf{ where } q \in s, s \in Q_d$$

*Since computations in the equivalent NFTA and DFTA will halt in (a set containing) the same state, a language is accepted in both if that state is a accepting state.*

---

**Proof 3.27.** *We will now prove that:*

$$\forall t \forall q \in Q_n (t \xrightarrow[\mathcal{A}]{*} q(t)) \implies \exists s \in Q_d (t \xrightarrow[\mathcal{A}_d]{*} s(t)), \textbf{ with } q \in s$$

*by using induction on terms.*

***Basis:*** *Consider $t \in \mathcal{F}_0$ ($t$ is a constant). If there is a transition $t \xrightarrow[\mathcal{A}]{} q(t)$, then according to the algorithm there is a transition $t \xrightarrow[\mathcal{A}_d]{} s(t)$ with $q \in s$.*

***Inductive step:*** *Consider $f \in \mathcal{F}_n$, where $n > 0$. We should then prove that if there is a transition*
*$f(t_1, \ldots, t_n) \xrightarrow[\mathcal{A}]{*} f(q_1(t_1), \ldots, q_n(t_n)) \to q(f(t_1, \ldots, t_n))$ in the NFTA,*

*we have a equivalent transition in the DFTA*
*$f(t_1, \ldots, t_n) \xrightarrow[\mathcal{A}_d]{*} f(s_1(t_1), \ldots, s_n(t_n)) \to s(f(t_1, \ldots, t_n))$*

***with*** $q \in s, q_1 \in s_1, \ldots, q_n \in s_n$

*We have* $\forall_{i \in 1,\ldots,n} t_i \xrightarrow[\mathcal{A}_d]{*} s_i(t_i)$ *with* $q_i \in s_i$ *as Induction Hypothesis.*

*In* $\mathcal{A}$ *we have a rule* $f(q_1(t_1) \ldots q_n(t_n)) \underset{\mathcal{A}}{\longrightarrow} q(f(t_1, \ldots, t_n)$ *Applying the algorithm we will have a rule* $f(s_1(t_1), \ldots s_n(t_n)) \underset{\mathcal{A}_d}{\longrightarrow} s(f(t_1, \ldots, t_n))$ *for some* $s$ *with* $q \in s$*, resulting in* $f(t_1, \ldots, t_n) \xrightarrow[\mathcal{A}_d]{*} f(s_1(t_1), \ldots, s_n(t_n)) \rightarrow s(f(t_1, \ldots, t_n))$
***with*** $q \in s, q_1 \in s_1, \ldots, q_n \in s_n$.

---

**Proof 3.28.** *We will now prove that:*

$$\forall t \forall s \in Q_d (t \xrightarrow[\mathcal{A}_d]{*} s(t)) \implies \exists q \in s (t \xrightarrow[\mathcal{A}]{*} q(t))$$

*by using induction on terms.*

***Basis:*** *Consider* $t \in \mathcal{F}_0$ *(*$t$ *is a constant). If there is a transition* $t \underset{\mathcal{A}_d}{\longrightarrow} s(t)$*, then according to the algorithm there exist* $\forall_{q \in s} t \underset{\mathcal{A}}{\longrightarrow} q(t)$.

***Inductive step:*** *Consider* $f \in \mathcal{F}_n$*, where* $n > 0$*. We should then prove that if there is a transition*
$f(t_1, \ldots, t_n) \xrightarrow[\mathcal{A}_d]{*} f(s_1(t_1), \ldots, s_n(t_n)) \rightarrow s(f(t_1, \ldots, t_n))$
*we have a equivalent transition in the NFTA*

$f(t_1, \ldots, t_n) \xrightarrow[\mathcal{A}]{*} f(q_1(t_1), \ldots, q_n(t_n)) \rightarrow q(f(t_1, \ldots, t_n))$,

***with*** $q \in s, q_1 \in s_1, \ldots, q_n \in s_n$

*We have* $\forall_{i \in 1,\ldots,n} t_i \xrightarrow[\mathcal{A}]{*} q_i(t_i)$ *with* $q_i \in s_i$ *as Induction Hypothesis.*

*Following the algorithm a rule* $f(s_1(t_1), \ldots s_n(t_n)) \underset{\mathcal{A}_d}{\longrightarrow} s(f(t_1, \ldots, t_n))$ *is made if there originally was a rule* $f(q_1(t_1), \ldots q_n(t_n)) \underset{\mathcal{A}}{\longrightarrow} q(f(t_1, \ldots, t_n))$ ***with*** *for every* $q$ *in* $s$*,* ***with*** $\exists_{q_1,\ldots,q_n} q_1 \in s_1, \ldots, q_n \in s_n$.

---

## 3.5 Reduction

Tree automata, much like regular automata, allow for unaccessible states to exist. Since these states are unaccessible, they do not determine whether a language is accepted or have any other major influence on the workings of the automaton. Therefore a reduction algorithm exists to remove these unaccessible states, which was proposed in TATA (p25).

**Algorithm 3.29.** *Reduction Algorithm*
*RED (NFTA* $\mathcal{A} = (Q, \mathcal{F}, Q_f, \Delta)$*)*

$M = \emptyset$
*While (States can be added to M)*
{
$M = M \cup q$
***where*** $f \in \mathcal{F}, q_1, \ldots, q_n \in M$,
$f(q_1(x_1), \ldots, q_n(x_n)) \to q(f(x_1, \ldots x_n)) \in \Delta$
}
$Q_r = M$
$Q_{rf} = Q_f \cap M$
$\Delta_r = \{f(q_1(x_1), \ldots, q_n(x_n)) \to q(f(x_1, \ldots, x_n)) \in \Delta | q, q_1, \ldots, q_n \in M\}$
***return*** *NFTA* $\mathcal{A}_r = (Q_r, \mathcal{F}, Q_{rf}, \Delta_r)$

---

**Example 3.30.** *Given a tree automaton for which:*
$\mathcal{F} = \{f(), a\}$, $\{q_1, q_2, q_f\} \in Q$,$\{q_f\} \in Q_f$ *and* $\{a \to q_f(a), f(q_1(x)) \to q_2(f(x))\} \in \Delta$

*First* $q_f$ *is added to* $M$*, since there are no states currently in* $M$ *and the rule* $a \to q_f(a)$ *has no need for any states to be found.*

*The other states can not be added to* $M$*, since they do not satisfy the requirements for being added. In the case of* $q_1$ *there exists no rule that has* $q_1$ *on the right hand side. In the case of* $q_2$ *there is a rule but it requires a state* $q_1$ *as the argument of* $f()$*, and since* $q_1 \notin M, q_2$ *can not be added either.*

$Q_r$ *is then created using everything in* $M = \{q_f\}$.$Q_{rf}$ *is then made consisting of only* $q_f$*, since this is the only state in both* $Q_f$ *and* $M$.$\Delta_r$ *is then set to be* $\{a \to q_f(a)\}$*, since it is the only rule in* $\Delta$ *whose states are in the list of accessible states.*

*The NFTA* $A_r = (Q_r, \mathcal{F}, Q_{rf}, \Delta_r)$ *is then returned.*

---

**Example 3.31.** *We define a tree automaton with*
$\mathcal{F} = \{f(), a\}$, $\{q_1, q_2, q_3, q_4, q_5, q_f\} \in Q$,$\{q_5, q_f\} \in Q_f$ *and* $\{a \to q_1(a), f(q_1(x)) \to q_2(f(x)), f(q_3(x)) \to q_4(f(x)), f(q_2(x)) \to q_f(f(x))\} \in \Delta$

| | $M$ | *Rule* |
|---|---|---|
| *0* | | |
| *1* | $q_1$ | $a \to q_1(a)$ |
| *2* | $q_2$ | $f(q_1(x)) \to q_2(f(x))$ |
| *3* | $q_f$ | $f(q_2(x)) \to q_f(f(x))$ |

*This will result in a new NFTA with a set* $Q_r$ *which contains all the items in the column labeled M, a* $\Delta_r$ *containing all items in the rule column,* $Q_{rf} = \{q_f\}$*. The resulting NFTA has only accessible states, therefore* $q_3, q_4$ *and* $q_5$ *have been removed.*

---

## 3.6 NFTA with epsilon rules

We briefly mentioned the NFTA-$\varepsilon$ in an earlier section. NFTA's can be extended with $\varepsilon$ rules, which are similar to the NFA's $\lambda$ rules. Using $\varepsilon$ rules a NFTA can change a state without taking any input.

**Definition 3.32.** *A NFTA-$\varepsilon$ is a Tree automaton that has transitions rules of the form:*

$$f(q_1(x_1), \ldots, q_n(x_n)) \to q_x(f(x_1, \ldots, x_n))$$

*and $\epsilon$ rules of the form:*

$$q(x) \to q'(x)$$

For any NFTA-$\varepsilon \mathcal{A}_\epsilon$ there is a NFTA $\mathcal{A}$ that accepts the same language, thus $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}_\epsilon)$; There exists no NFTA-$\varepsilon$ that accepts a language that cannot be accepted by any NFTA. Thus the NFTA-$\varepsilon$ only exists to simplify proofs and examples.

We will now introduce a new function:

**Definition 3.33.** *$\varepsilon close(q)$ is the set of states that are accessible from $q$ by only using $\varepsilon$ rules. It is inductively defined as:*

- $q \in \varepsilon close(q)$
- $q'' \in \varepsilon close(q)$ ***if*** $q' \in \varepsilon close(q)$ ***and*** $q'(x) \to q''(x) \in \Delta$

---

**Example 3.34.** *Given states $q_1, q_2, q_3, q_4$, and $\Delta = \{q_1(x) \to q_2(x), q_2(x) \to q_3(x), q_4(x) \to q_1(x)\}$, we get:*
$\varepsilon close(q_1) = \{q_1, q_2, q_3\}$
$\varepsilon close(q_2) = \{q_2, q_3\}$
$\varepsilon close(q_3) = \{q_3\}$
$\varepsilon close(q_4) = \{q_1, q_2, q_3, q_4\}$

---

Though TATA states that all NFTA-$\varepsilon$ can be transformed into an equivalent NFTA, it doesn't propose any algorithms to do so. Therefore we introduce the following algorithm:

**Algorithm 3.35.** *Epsilon removal Algorithm*
***EPS*** *(NFTA-$\varepsilon$ $\mathcal{A} = (Q_e, \mathcal{F}, Q_{ef}, \Delta_e)$)*

$Q = \emptyset$
$\Delta = \emptyset$
***While*** *(rules can be added to $\Delta_d$)*
{
$Q = Q \cup \{s\}$
$\Delta = \Delta \cup \{f(s_1(x), \ldots, s_n(x)) \to s\}$
***where*** $f \in \mathcal{F}, \quad s_1, \ldots, s_n \in Q$
$s = \{\varepsilon close(q) | \exists (q_1 \in s_1, \ldots, q_n \in s_n),$

$f(q_1(x_1), \ldots, q_n(x_n)) \to q(f(x_1, \ldots x_n)) \in \Delta_e\}$
***and*** $s \neq \emptyset$

}
$Q_f = \{s \in Q | s \cap Q_{ef} \neq \emptyset\}$
***return*** *NFTA* $\mathcal{A} = (Q, \mathcal{F}, Q_f, \Delta)$

---

**Example 3.36.** *Given an NFTA-*$\varepsilon$ $\mathcal{A}_e = (Q_e, \mathcal{F}, Q_{ef}, \Delta_e)$ *with*
$Q_e = \{q_a, q_b, q_c, q_f\}$, $\mathcal{F} = \{a, f()\}$ *and*
$\Delta_e = \{a \to q_a(a), q_a(x) \to q_b(x), f(q_a(x)) \to q_c(f(x)), f(q_b(x)) \to q_f(f(x))\}$

*Using the Epsilon removal algorithm, we get the following table of steps:*

| | $\{s\}$ | *Rule* |
|---|---|---|
| *0* | | |
| *1* | $\{q_a, q_b\}$ | $a \to \{q_a, q_b\}(a)$ |
| *2* | $\{q_c\}$ | $f(\{q_a, q_b\}(x)) \to \{q_c\}(f(x))$ |
| *3* | $\{q_f\}$ | $f(\{q_a, q_b\}(x)) \to \{q_f\}(f(x))$ |

*In the first step the only rule we can add is the rule that states* $a \to q_a(a)$*, and since there is an* $\epsilon$ *rule that moves from* $q_a$ *to* $q_b$*, that rule is altered to reach the set of both states.*

*We then have two choices of rules to add, as presented in steps 2 and 3. In step 2 we start with the original rule* $f(q_a(x)) \to q_c(f(x))$*. From this rule we can only derive one rule, since the only set of states in* $Q$ *containing* $q_a$ *is* $\{q_a, q_b\}$*, thus creating* $f(\{q_a, q_b\}(x)) \to \{q_c\}(f(x))$*.*

*Step 3 does the same thing for the rule* $f(q_b(x)) \to q_f(f(x))$*, resulting in the rule* $f(\{q_a, q_b\}(x)) \to \{q_f\}(f(x))$ *.*

*Finally all sets of states containing states that are in* $Q_{ef}$ *are put in the set of final states* $Q_f$ *for the newly created NFTA.*

---

**Example 3.37.** *Given an NFTA-*$\varepsilon$ $\mathcal{A}_e = (Q_e, \mathcal{F}, Q_{ef}, \Delta_e)$ *with*
$Q_e = \{q_a, q_b, q_c, q_e, q_f, q_g\}$,$Q_{ef} = \{q_a, q_f\}$, $\mathcal{F} = \{a, b, f(), g(,)\}$ *and* $\Delta_e$ *shown in the table below:*

| *Rule* | $\varepsilon$*Rule* |
|---|---|
| $a \to q_a(a)$ | $q_a(x) \to q_b(x)$ |
| $b \to q_b(b)$ | $q_b(x) \to q_c(x)$ |
| $f(q_a(x)) \to q_f(f(x))$ | $q_g(x) \to q_a(x)$ |
| $f(q_c(x)) \to q_e(f(x))$ | |
| $g(q_e(x), q_f(y)) \to q_g(g(x, y))$ | |

*Following the algorithm, we get the table below:*

| | $\{s\}$ | *Rule* |
|---|---|---|
| *0* | | |
| *1* | $\{q_a, q_b, q_c\}$ | $a \to \{q_a, q_b, q_c\}(a)$ |
| *2* | $\{q_b, q_c\}$ | $b \to \{q_b, q_c\}(b)$ |
| *3* | $\{q_f\}$ | $f(\{q_a, q_b, q_c\}(x)) \to \{q_f\}(f(x))$ |
| *4* | $\{q_e\}$ | $f(\{q_a, q_b, q_c\}(x)) \to \{q_e\}(f(x))$ |
| *5* | $\{q_e\}$ | $f(\{q_b, q_c\}(x)) \to \{q_e\}(f(x))$ |
| *6* | $\{q_a, q_b, q_c, q_g\}$ | $g(\{q_e\}(x), \{q_f\}(y)) \to \{q_a, q_b, q_c, q_g\}(g(x, y))$ |
| *7* | $\{q_f\}$ | $f(\{q_a, q_b, q_c, q_g\}(x)) \to \{q_f\}(f(x))$ |
| *8* | $\{q_e\}$ | $f(\{q_a, q_b, q_c, q_g\}(x)) \to \{q_e\}(f(x))$ |

*This will result in a new NFTA with a set $Q$ which contains all the items in the column labeled $\{s\}$, a $\Delta$ containing all items in the rule column, $Q_f = \{\{q_a, q_b, q_c\}, \{q_f\}, \{q_a, q_b, q_c, q_g\}\}$.*

---

**Proof 3.38.** *We will now prove that:*

$$\forall t \forall q \in Q_e(t \xrightarrow[\mathcal{A}_e]{*} q(t)) \implies \exists s \in Q(t \xrightarrow[\mathcal{A}]{*} s(t)), \textbf{\textit{with}}\ q \in s$$

*by using induction on terms.*

***Basis:*** *Consider $t \in \mathcal{F}_0$. If there is a transition $t \xrightarrow[\mathcal{A}_e]{} q(t)$, then according to the algorithm there is a transition $t \xrightarrow[\mathcal{A}]{} s(t)$ with $q \in s$.*

***Inductive step:*** *Consider $f \in \mathcal{F}_n$, where $n > 0$. We should then prove that if there is a transition*
*$f(t_1, \ldots, t_n) \xrightarrow[\mathcal{A}_e]{*} f(q_1(t_1), \ldots, q_n(t_n)) \to q(f(t_1, \ldots, t_n))$ in the NFTA-$\varepsilon$,*

*we have a equivalent transition in the NFTA*
*$f(t_1, \ldots, t_n) \xrightarrow[\mathcal{A}]{*} f(s_1(t_1), \ldots, s_n(t_n)) \to s(f(t_1, \ldots, t_n))$*
***with*** *$q \in s, q_1 \in s_1, \ldots, q_n \in s_n$*

*We have $\forall_{i \in 1,\ldots,n} t_i \xrightarrow[\mathcal{A}]{*} s_i(t_i)$ with $q_i \in s_i$ as Induction Hypothesis.*

*In $\mathcal{A}_e$ we have a rule $f(q_1(t_1) \ldots q_n(t_n)) \xrightarrow[\mathcal{A}]{} q(f(t_1, \ldots, t_n)$ Applying the algorithm we will have a rule $f(s_1(t_1), \ldots s_n(t_n)) \xrightarrow[\mathcal{A}]{} s(f(t_1, \ldots, t_n))$ for some $s$ with $q \in s$, resulting in $f(t_1, \ldots, t_n) \xrightarrow[\mathcal{A}]{*} f(s_1(t_1), \ldots, s_n(t_n)) \to s(f(t_1, \ldots, t_n))$*
***with*** *$q \in s, q_1 \in s_1, \ldots, q_n \in s_n$*

---

**Proof 3.39.** *We will now prove that:*

$$\forall t \forall s \in Q(t \xrightarrow[\mathcal{A}]{*} s(t)) \implies \exists q \in s(t \xrightarrow[\mathcal{A}_e]{*} q(t))$$

*by using induction on terms.*

***Basis:*** *Consider $t \in \mathcal{F}_0$ (t is a constant). If there is a transition $t \xrightarrow[\mathcal{A}]{} s(t)$, then according to the algorithm there exist $\forall_{q \in s} t \xrightarrow[\mathcal{A}_e]{} q(t)$.*

***Inductive step:*** *Consider $f \in \mathcal{F}_n$, where $n > 0$. We should then prove that if there is a transition*
$f(t_1, \ldots, t_n) \xrightarrow[\mathcal{A}]{*} f(s_1(t_1), \ldots, s_n(t_n)) \to s(f(t_1, \ldots, t_n))$
*we have a equivalent transition in the NFTA-$\varepsilon$*

$$f(t_1, \ldots, t_n) \xrightarrow[\mathcal{A}_e]{*} f(q_1(t_1), \ldots, q_n(t_n)) \to q(f(t_1, \ldots, t_n))$$

***with*** $q \in s, q_1 \in s_1, \ldots, q_n \in s_n$

*We have $\forall_{i \in 1,\ldots,n} t_i \xrightarrow[\mathcal{A}]{*} q_i(t_i)$ with $q_i \in s_i$ as Induction Hypothesis.*

*Following the algorithm a rule $f(s_1(t_1), \ldots s_n(t_n)) \xrightarrow[\mathcal{A}]{} s(f(t_1, \ldots, t_n))$ is made if there originally was a rule $f(q_1(t_1), \ldots q_n(t_n)) \xrightarrow[\mathcal{A}_e]{} q(f(t_1, \ldots, t_n))$ **with** for every $q$ in $s$, **with** $\exists_{q_1,\ldots,q_n} q_1 \in s_1, \ldots, q_n \in s_n$*

---

# 4 Language properties for Tree automata

## 4.1 Pumping Lemma for Tree Automata

In Section 1.2 of TATA[1] the pumping lemma is given for tree automata:

**Theorem 4.1.** *Let $L$ be a recognizable set of ground terms. Then there exists a $k > 0$ satisfying:*

*For every ground term $t$ in $L$ such that $height(t) > k$ there exists:*

1. *a context $C \in K(\mathcal{F})$*
2. *a nontrivial context $C' \in K(\mathcal{F})$*
3. *a ground term $u$*

   *such that $t = C[C'[u]]$ and $\forall n \geq 0, C[C'^n[u]] \in L$*

---

**Example 4.2.** *Given $\mathcal{F} = \{a, f(), g()\})$ and the recognizable language*

$$L = \{f(g^i(a))|i \geq 0\}$$

*In this case we can choose $k = 2$, $C = f(x), C' = g(x)$, $u = a$, and we observe that $\forall_n C[C'^n[u]] \in L$.*

---

**Example 4.3.** *Given $\mathcal{F} = \{a, f(), g()\})$ and a recognizable language*

$$L = \{g((f(g(f)))^i(a)))|i > 0\}$$

*Which means that, after an initial $g()$, the segment $f(g(f()))$ is repeated $i$ times followed by an $a$. Examples are $g(f(g(f(a))))$ and $g(f(g(f(f(g(f(a)))))))$.*

*In this case we can choose $k = 2$, $C = g(x), C' = f(g(f(x)))$, $u = a$, and we observe that $\forall_n C[C'^n[u]] \in L$.*

---

**Example 4.4.** *Given $\mathcal{F} = \{a, f(,), g()\})$ and a language*
*$L = \{f(g^i(a), (g^i(a))|i \geq 0\}$.*

*Suppose $L$ is recognizable, then according to Theorem 4.1, there is a $k > 0$ such that the properties in 4.1 hold.*

*Take $t = f(g^k(a), g^k(a))$, then $height(t) > k$. So there is a context $C$ and nontrivial context $C'$ and a $u$ such that $t = C[C'[u]]$ and $\forall_n C[C'^n[u]] \in L$*

*Now there are the following possibilities choices for $C, C'$ and $u$:*

- *$C = f(x, g^k(a)), C' = g^m(x), u = g^j(a), j + m = k, j < k$ : Take $n = 0$, resulting in $f(g^k(a), g^j(a)) \in L$.*

- $C = f(g^k(a), x), C' = g^m(x), u = g^j(a), j + m = k, j < k$ : *Take* $n = 0$, *resulting in* $f(g^k(a), g^j(a)) \in L$.
- $C = x, C' = f((x, g^k(a))), u = g^k(a)$ : *Take* $n = 0$, *resulting in* $g^k(a) \in L$.
- $C = x, C' = f((g^k(a)), x), u = g^k(a)$ : *Take* $n = 0$, *resulting in* $g^k(a) \in L$.

*All these options lead towards a contradiction , because the resulting word is not in L, therefore L is not recognizable.*

---

**Proof of** (4.1)**.** *Let* $\mathcal{A} = (Q, \mathcal{F}, Q_f, \Delta)$ *be a FTA such that* $L = \mathcal{L}(a)$, *and* $k = |Q|$. *Consider a ground term* $t \in L$ *such that* $height(t) > k$.

*Consider a successful run* $r$ *on* $t$ *by* $\mathcal{A}$. *Since* $height(t) > k$ *and* $k = |Q|$, *there is a path in* $t$ *that is longer than* $k$ *and there are 2 positions* $p_1 < p_2$ *such that* $r(p_1) = r(p_2) = q$ *for some state* $q$.

*Let* $u_1$ *be the ground subterm of* $t$ *at* $p_1$, *and* $u$ *the ground subterm of* $t$ *at* $p_2$. *There then exists a non-trivial context* $C'$ *such that* $u_1 = C'[u]$. *We then define the context* $C$ *such that* $t = C[C'[u]]$.

*Consider a term* $t = C[C'^n[u]]$ *for some* $n > 1$. *Suppose that* $r$ *corresponds to the reduction* $t \xrightarrow[\mathcal{A}]{*} q_f, q_f \in Q_f$ *then we have:*

$$C[C'^n[u]] \xrightarrow[\mathcal{A}]{*} C[C'^n[q]] \xrightarrow[\mathcal{A}]{*} C[C'^{n-1}[q]] \xrightarrow[\mathcal{A}]{*} \dots C[q] \xrightarrow[\mathcal{A}]{*} q_f.$$

*The same holds for* $n = 0$

---

## 4.2 Closure properties

For the class of regular languages over words there are several closure properties that hold, such as Union and Intersection. Chapter 1.3 of TATA[1] describes that these properties hold for the class of regular tree languages as well, and gives the theory for these properties, but no proof nor any examples. This section will therefore give a more complete view on these closure properties.

Regular tree languages are the class of languages which are accepted by a NFTA, and therefore by any other previously mentioned type of tree automaton. Like in for the class of regular languages over words, there exist several closure properties for the class of the class of regular tree languages. These include **union**, **complementation** and **intersection**; All of which will be described and proven in the coming subsections.

We will prove these closure properties by construction of automata, since regular tree languages consist of languages accepted by a NFTA we can prove these properties by showing that automata for the languages created by these closure properties of tree regular tree languages can be made.

### 4.2.1 Union

The first closure property we will describe is **union**, however we must first note that there are 2 possible approaches to this property. We will start with the simpler version, which does not preserve the determinism of the tree languages that are put in union.

**Definition 4.5.** *Given* $\mathcal{A}_1 = (Q_1, \mathcal{F}, Q_{1f}, \Delta_1)$ *and* $\mathcal{A}_2 = (Q_2, \mathcal{F}, Q_{2f}, \Delta_2)$ *with* $Q_1 \cap Q_2 = \emptyset$ *(which can also be enforced by renaming states), the tree automaton* $\mathcal{A}_1 \cup \mathcal{A}_2$ *is defined as follows.* $\mathcal{A} = (Q, \mathcal{F}, Q_f, \Delta)$ *with:*

- $Q = Q_1 \cup Q_2$
- $Q_f = Q_{1f} \cup Q_{2f}$
- $\Delta = \Delta_1 \cup \Delta_2$

**Theorem 4.6.** *Let* $L_1$ *and* $L_2$ *be two recognizable tree languages for which* $L_1 = \mathcal{L}(\mathcal{A}_1)$ *and* $L_2 = \mathcal{L}(\mathcal{A}_2)$*. Then,* $L_1 \cup L_2 = \mathcal{L}(\mathcal{A}_1) \cup \mathcal{L}(\mathcal{A}_2)$

Note that by using Definition 4.5 and Theorem 4.6, the properties of determinism and completeness for $\mathcal{A}_1$ and $\mathcal{A}_2$ are not preserved in $\mathcal{A}$.

---

**Example 4.7.** *Given* $\mathcal{A}_1 = (Q_1, \mathcal{F}, Q_1, \Delta_{f1})$ *with* $\mathcal{F} = \{a, f()\}$, $Q_1 = \{q_1, q_2, q_3\}$, $Q_{f1} = \{q_1, q_3\}$ *and* $\Delta_1 = \{a \to q_1(a), f(q_1(x)) \to q_3(f(x)), f(q_3(x) \to q_2(f(x))\}$*; and* $\mathcal{A}_2 = (Q_2, \mathcal{F}, Q_2, \Delta_{f2})$*, with* $Q_2 = \{q_a, q_b, q_c\}$, $Q_{f2} = \{q_c\}$ *and* $\Delta_2 = \{a \to q_a(a), f(q_a(x)) \to q_b(f(x)), f(q_b(x) \to q_c(f(x)), f(q_c(x) \to q_c(f(x))\}$

*We can see that* $L_1 = \{f(a) \vee a\}$ *and* $L_2 = \{f^n(a) | n \geq 2\}$*. Now, according to Theorem 4.6* $L = L_1 \cup L_2 = \{f(a) \vee a \vee f^n(a) | n \geq 2\} = \{f^i(a) | a \geq 0\}$.

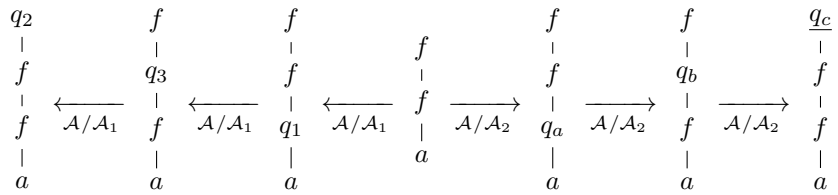
*Now let us consider* $t_1 = a$, $t_2 = f(a)$ *and* $t_3 = f(f(a))$*. We shall now show how these trees are processed in* $\mathcal{A}$, $\mathcal{A}_1$ *and* $\mathcal{A}_2$*. We start with* $t_1$*:*

$$\underline{q_1} \quad \xleftarrow[\mathcal{A}/\mathcal{A}_1]{} \quad a \quad \xrightarrow[\mathcal{A}/\mathcal{A}_2]{} \quad q_a$$
$$\vert \qquad\qquad\qquad\qquad\qquad\qquad \vert$$
$$a \qquad\qquad\qquad\qquad\qquad\qquad a$$

*In which we underlined* $q_1$ *to mark it is an accepting state. we can see that* $\mathcal{A}$ *accepts* $t_1$ *due to a rule it received from* $\mathcal{A}_1$*, yet it can also apply a rule from* $\mathcal{A}_2$ *that leads to a state that is not accepted. For* $t_2$ *the following transitions exist:*

$$\underline{q_3} \quad \xleftarrow[\mathcal{A}/\mathcal{A}_1]{} \quad f \quad \xleftarrow[\mathcal{A}/\mathcal{A}_1]{} \quad f \quad \xrightarrow[\mathcal{A}/\mathcal{A}_2]{} \quad f \quad \xrightarrow[\mathcal{A}/\mathcal{A}_2]{} \quad q_b$$
$$\vert \qquad\qquad \vert \qquad\qquad \vert \qquad\qquad \vert \qquad\qquad \vert$$
$$f \qquad\qquad q_1 \qquad\qquad \vert \qquad\qquad q_a \qquad\qquad a$$
$$\vert \qquad\qquad \vert \qquad\qquad a \qquad\qquad \vert \qquad\qquad \vert$$
$$a \qquad\qquad a \qquad\qquad\qquad\qquad a \qquad\qquad a$$

*We can see that $t_2$ is accepted for the same reasons as $t_1$. $t_3$ however was not in $L_1$, but since it was in $L_2$ it will still be accepted by $\mathcal{A}$, using the rules it received from $\mathcal{A}_2$:*




*We can see in these examples even though both $\mathcal{A}_1$ and $\mathcal{A}_2$ were deterministic, the unified machine is nondeterministic. Also note that due to $Q_1 \cap Q_2 = \emptyset$, once the first reduction in one direction is made, no rules applied in the other direction can be used.*

---

**Proof of (4.6).** *Since $\mathcal{L}(\mathcal{A}_1) \cup \mathcal{L}(\mathcal{A}_2)$ uses the transition rules from both $\mathcal{A}_1$ and $\mathcal{A}_2$, which cannot access the same states, therefore it only accepts trees that are in the language of either $\mathcal{A}_1$, $\mathcal{A}_2$ or both, and nothing more.*

*Therefore if we have this machine for which $\mathcal{L}(\mathcal{A}) = L$, it follows that if $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}_1) \cup \mathcal{L}(\mathcal{A}_2)$, we have $L = \mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}_1) \cup \mathcal{L}(\mathcal{A}_2) = L_1 \cup L_2$.*

---

### 4.2.2 Union with preservation of determinism

In the last subsection we discussed a theorem for union that does not preserve determinism. Yet in TATA [1] there is a second theorem for union that does preserve determinism. The idea behind this theorem is that the two tree automata process the input term in parallel.

**Definition 4.8.** *Let $L_1$ and $L_2$ be two recognizable tree languages, and $\mathcal{A}_1 = (Q_1, \mathcal{F}.Q_{f1}, \Delta_1)$ and $\mathcal{A}_2 = (Q_2, \mathcal{F}.Q_{f2}, \Delta_2)$ be two tree automata for which $L_1 = \mathcal{L}(\mathcal{A}_1)$ and $L_2 = \mathcal{L}(\mathcal{A}_2)$.*

*Suppose that $Q_1 \cap Q_2 = \emptyset$, and that $\mathcal{A}_1$ and $\mathcal{A}_2$ are complete. $\mathcal{A}_1 \times \mathcal{A}_2$ is then defined as $\mathcal{A} = (Q, \mathcal{F}, Q, \Delta)$ with:*

- $Q = Q_1 \times Q_2$
- $Q_f = (Q_{f1} \times Q_2) \cup (Q_{f2} \times Q_1)$
- $\Delta = \Delta_1 \times \Delta_2$

  *with:* $\Delta_1 \times \Delta_2 = \{f((q_1, q_1')(x_1)), \ldots, f((q_n, q_n')(x_n)) \to (q, q')(f(x_1, \ldots x_n)) |$

  $f(q_1(x_1), \ldots, q_n(x_n)) \to q(f(x_1, \ldots, x_n)) \in \Delta_1,$

  $f(q_1'(x_1), \ldots, q_n'(x_n)) \to q'(f(x_1, \ldots, x_n)) \in \Delta_2 \quad \}$

**Theorem 4.9.** *Let $L_1$ and $L_2$ be two recognizable tree languages for which $L_1 = \mathcal{L}(\mathcal{A}_1)$ and $L_2 = \mathcal{L}(\mathcal{A}_2)$. Then, $L_1 \cup L_2 = \mathcal{L}(\mathcal{A}_1 \times \mathcal{A}_2)$, where $\mathcal{A}_1$ and $\mathcal{A}_2$ are two complete DFTA's.*

---

**Example 4.10.** *Given $\mathcal{F} = \{a, f(), g()\}$, $\mathcal{A}_1 = (Q_1, \mathcal{F}, Q_1, \Delta_{1f})$ with:*

- $Q_1 = \{q_1, q_2\}$
- $Q_{1f} = \{q_2\}$
- $\Delta_1 = \{a \to q_1(a), f(q_1(x)) \to q_2(f(x)), f(q_2(x)) \to q_2(f(x)), g(q_1(x)) \to q_1(g(x)), g(q_2(x)) \to q_2(g(x))\}$

*And $\mathcal{A}_2 = (Q_2, \mathcal{F}, Q_2, \Delta_{2f})$, with:*

- $Q_2 = \{q_a, q_b\}$
- $Q_{2f} = \{q_b\}$
- $\Delta_2 = \{a \to q_a(a), f(q_a(x)) \to q_a(f(x)), f(q_b(x)) \to q_b(f(x)), g(q_a(x)) \to q_b(g(x)), g(q_b(x)) \to q_b(g(x))\}$

*This means that means $\mathcal{A}_1$ accepts any tree that contains at least one $f$, $\mathcal{A}_2$ does the same with $g$. We create a new machine $\mathcal{A} = \mathcal{A}_1 \times \mathcal{A}_2$. Applying the rules in 4.8, we get $\mathcal{A} = (Q, \mathcal{F}, Q_f, \Delta)$ with:*

- $Q = \{\{q_1, q_a\}, \{q_1, q_b\}, \{q_2, q_a\}, \{q_2, q_b\}\}$
- $Q_f = \{\{q_1, q_b\}, \{q_2, q_a\}, \{q_2, q_b\}\}$
- $\Delta = \{a \to (q_1, q_a)(a),$
  $f((q_1, q_a)(x)) \to (q_2, q_a)(f(x)),$
  $f((q_1, q_b)(x)) \to (q_2, q_b)(f(x)),$
  $f((q_2, q_a)(x)) \to (q_2, q_a)(f(x)),$
  $f((q_2, q_b)(x)) \to (q_2, q_b)(f(x)),$
  $g((q_1, q_a)(x)) \to (q_1, q_b)(g(x)),$
  $g((q_1, q_b)(x)) \to (q_1, q_b)(g(x)),$
  $g((q_2, q_a)(x)) \to (q_2, q_b)(g(x)),$
  $g((q_2, q_b)(x)) \to (q_2, q_b)(g(x))\}$

*Now let us consider the trees $t_1 = a, t_2 = f(a), t_3 = f(g(a))$ and $t_4 = g(a)$. First we will show the reduction of $t_1$:*

$$a \xrightarrow[\mathcal{A}]{} \begin{matrix}(q_1, q_a)\\ |\\ a\end{matrix}$$

*$t_1$ is accepted by neither $\mathcal{A}_1$ nor $\mathcal{A}_2$, and the resulting machine from $\mathcal{A}_1 \times \mathcal{A}_2$ does not accept it either. Now for $t_2$, which would have been accepted by $\mathcal{A}_1$:*

$$\begin{array}{ccccc} f & & f & & \underline{(q_2,q_a)} \\ | & \xrightarrow[\mathcal{A}]{} & (q_1,q_a) & \xrightarrow[\mathcal{A}]{} & f \\ a & & | & & | \\ & & a & & a \end{array}$$

*We get a machine with a final state above its tree after the run of our new machine, thus accepting $t_2$. Now we show a run on $t_3$:*

$$\begin{array}{ccccccc} f & & f & & f & & \underline{(q_2,q_b)} \\ | & & | & & | & & | \\ g & \xrightarrow[\mathcal{A}]{} & g & \xrightarrow[\mathcal{A}]{} & (q_1,q_b) & \xrightarrow[\mathcal{A}]{} & f \\ | & & | & & | & & | \\ a & & (q_1,q_a) & & g & & g \\ & & | & & | & & | \\ & & a & & a & & a \end{array}$$

*$t_3$ is accepted by both $\mathcal{A}_1$ and $\mathcal{A}_2$, which can be seen back in $\mathcal{A}$'s run on $t_3$ by its resulting final state on top being one containing final states from both machines it embeds. $t_3$ is therefore accepted by $\mathcal{A}$. Now for our final tree, $t_4$:*

$$\begin{array}{ccccc} g & & g & & \underline{(q_1,q_b)} \\ | & \xrightarrow[\mathcal{A}]{} & (q_1,q_a) & \xrightarrow[\mathcal{A}]{} & g \\ a & & | & & | \\ & & a & & a \end{array}$$

*$t_4$ is accepted by $\mathcal{A}_2$, and as it shows above it is accepted by $\mathcal{A}$ as well. The tuple $(q_1,q_b)$ also shows us that $\mathcal{A}_1$ would not have reached an accepting state, and would have rejected $t_4$.*

---

**Proof of (4.9).** *Let $L_1$ and $L_2$ be two recognizable tree languages, $\mathcal{A}_1 = (Q_1, \mathcal{F}.Q_{1f}, \Delta_1)$ and $\mathcal{A}_2 = (Q_2, \mathcal{F}.Q_{2f}, \Delta_2)$ be two complete DFTA's such that $L_1 = \mathcal{L}(\mathcal{A}_1)$ and $L_2 = \mathcal{L}(\mathcal{A}_2)$. Let $\mathcal{A} = \mathcal{A}_1 \times \mathcal{A}_2$ be a automaton constructed using Definition 4.8. Then there are 2 types of cases which could occur while $\mathcal{A}$ does a run on some tree $t$:*

$\nexists_{q\in Q_1} t \xrightarrow[\mathcal{A}_1]{*} q(t) \vee \nexists_{q'\in Q_2} t \xrightarrow[\mathcal{A}_2]{*} q'(t)$ :

*These cases in which either machine, or both, get stuck cannot occur since we are working with complete automata, which have transition rules for every possible input (3.18).*
*If one of these automata would not be complete however, this could cause the machine to get stuck in while processing a tree that would be accepted by the other machine, resulting in $\mathcal{A}$ getting stuck and rejecting that tree. Therefore it is imperative that both machines are complete.*

$\exists_{q\in Q_1, q'\in Q_2} t \xrightarrow[\mathcal{A}_1]{*} q(t) \wedge t \xrightarrow[\mathcal{A}_2]{*} q'(t)$ :

*In these cases the machine $\mathcal{A}$ will have a run $t \xrightarrow[\mathcal{A}]{*} (q,q')(t)$ in which the following cases can occur:*

$\underline{q \in Q_{1f} \vee q' \in Q_{2f}}$ : *From Definition 4.9 follows that if either of the states $q$ and $q'$ is an accepting state, the state $(q, q')$ is an accepting state. Therefore a tree $t$ is accepted by $\mathcal{A}$ if either $\mathcal{A}_1$ and/or $\mathcal{A}_2$ accepts this tree.*

$\underline{q \notin Q_{1f} \wedge q' \notin Q_{2f}}$ : *From Definition 4.9 follows that if neither of the states $q$ and $q'$ is an accepting state, the state $(q, q')$ will not be an accepting state either. Therefore a tree $t$ is rejected by $\mathcal{A}$ if both $\mathcal{A}_1$ and $\mathcal{A}_2$ machine reject this tree.*

*Therefore $\mathcal{A}$ accepts trees only if $\mathcal{A}_1$ and/or $\mathcal{A}_2$ accepts them, thus $L = \mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}_1) \times \mathcal{L}(\mathcal{A}_2) = L_1 \times L_2$.*

---

### 4.2.3 Complementation

**Definition 4.11.** *Let $L$ be a recognizable tree language, and $\mathcal{A} = (Q, \mathcal{F}.Q_f, \Delta)$ be a complete DFTA such that $L = \mathcal{L}(A)$. Now we can complement the set of final states to recognize the complement of $L$, creating an automaton $\mathcal{A}^c = (Q, \mathcal{F}.Q_f^c, \Delta)$ with:*

- $Q_f^c = Q \setminus Q_f$

To complement a NFTA, the determinization algorithm 3.23 and the completion algorithm 3.20 have to be run on it first. This may however lead to an exponential blow up.

**Theorem 4.12.** *Let $L$ be a recognizable tree language, and $\mathcal{A} = (Q, \mathcal{F}.Q_f, \Delta)$ be a complete DFTA such that $L = \mathcal{L}(a)$. $\mathcal{A}^c$ recognizes the complement of $L$ in $T(\mathcal{F})$.*

*The complement of $L$ is denoted by $\overline{L}$*

---

**Example 4.13.** *Going back to the language of $\mathcal{A}$ from Example 4.10, we can create a complementary machine to it using Definition 4.11, resulting in $\mathcal{A}^c = (Q, \mathcal{F}.Q_f^c, \Delta)$ with $Q_f^c = \{(q_1, q_a)\}$.*

*This language should only accept languages not containing $f$ or $g$, thus only accepting $a$. Since the transition rules for this machine have not changed, the reductions for $\mathcal{A}$ and $\mathcal{A}^c$ will be equal to each other. So any tree reduction resulting in $(q_1, q_a)(t_x)$ in Example 4.10 will be accepted, all other reductions will be rejected.*

*This will result in only $t_1 = a$ to be accepted, and anything containing $f$ or $g$ to be rejected, thus if $L^c = \mathcal{L}(\mathcal{A}^c)$ then $L^c = \overline{L}$.*

---

**Proof of (4.12).** *Let $L$ be a recognizable tree language, and $\mathcal{A} = (Q, \mathcal{F}.Q_f, \Delta)$ be a complete DFTA such that $L = \mathcal{L}(a)$. Now there are 2 possibilities for a tree not to be accepted:*

- $t \to q(t)$ *with* $q \notin Q_f$*: If the state put above the tree is not a final state, the tree is rejected.*
- $t \not\to q(t)$*: Can not occur in a complete DFTA (see Definition 3.18).*

*Therefore the complement of L,* $\overline{L}$ *will only exist of trees that do get a state put above a tree after a run of* $\mathcal{A}$*. Therefore* $\mathcal{A}^c$ *only has to accept trees topped with those states* $\mathcal{A}$ *rejects, and vice versa. Definition 4.11 enforces by it's definition of* $Q_f^c$*, thus resulting in* $\mathcal{L}(\mathcal{A}_c) = \overline{L}$

---

### 4.2.4 Intersection

**Definition 4.14.** *Let* $L_1$ *and* $L_2$ *be two recognizable tree languages, and* $\mathcal{A}_1 = (Q_1, \mathcal{F}.Q_{f1}, \Delta_1)$ *and* $\mathcal{A}_2 = (Q_2, \mathcal{F}.Q_{f2}, \Delta_2)$ *be two tree automata for which* $L_1 = \mathcal{L}(\mathcal{A}_1)$ *and* $L_2 = \mathcal{L}(\mathcal{A}_2)$*.*

$\mathcal{A}_1 \cap \mathcal{A}_2$ *is then defined as* $\mathcal{A} = (Q, \mathcal{F}, Q, \Delta)$ *with:*

- $Q = Q_1 \times Q_2$
- $Q_f = Q_{f1} \times Q_{f2}$
- $\Delta = \Delta_1 \times \Delta_2$

  *with:* $\Delta_1 \times \Delta_2 = \{f((q_1, q_1'), \ldots, f((q_n, q_n')) \to (q, q') |$

  $f(q_1, \ldots, q_n) \to q \in \Delta_1, f(q_1', \ldots, q_n') \to q' \in \Delta_2\}$

**Theorem 4.15.** *Let* $L_1$ *and* $L_2$ *be two recognizable tree languages for which* $L_1 = \mathcal{L}(\mathcal{A}_1)$ *and* $L_2 = \mathcal{L}(\mathcal{A}_2)$*. Then,* $L_1 \cap L_2 = \mathcal{L}(\mathcal{A}_1 \cap \mathcal{A}_2)$

---

**Example 4.16.** *Taking the automata from Example 4.10, and creating a new machine* $\mathcal{A}_i = \mathcal{A}_1 \cap \mathcal{A}_2 = (Q, \mathcal{F}, Q_{fx}, \Delta)$*. The reductions for* $t_1, \ldots, t_4$ *would remain the same as in Example 4.10, yet since* $Q_{fx} = \{(q_2, q_b)\}$ *only* $t_3$ *would be accepted on this machine.*

---

**Proof of (4.15).** *Let* $L_1$ *and* $L_2$ *be two recognizable tree languages,* $\mathcal{A}_1 = (Q_1, \mathcal{F}.Q_{1f}, \Delta_1)$ *and* $\mathcal{A}_2 = (Q_2, \mathcal{F}.Q_{2f}, \Delta_2)$ *be two FTA's such that* $L_1 = \mathcal{L}(\mathcal{A}_1)$ *and* $L_2 = \mathcal{L}(\mathcal{A}_2)$*. Let* $\mathcal{A} = \mathcal{A}_1 \cap \mathcal{A}_2$ *be a automaton constructed using Definition 4.14. Then there are 2 types of cases which could occur while* $\mathcal{A}$ *does a run on some tree t:*

$\nexists_{q \in Q_1} t \xrightarrow[\mathcal{A}_1]{*} q(t) \vee \nexists_{q' \in Q_2} t \xrightarrow[\mathcal{A}_2]{*} q'(t)$ :

*In these cases that either machine* $\mathcal{A}_1$ *or* $\mathcal{A}_2$ *gets stuck, the automaton* $\mathcal{A}$ *gets stuck as well and rejects the tree, just like* $\mathcal{A}_1$ *or* $\mathcal{A}_2$ *would have rejected that tree.*

$\underline{\exists_{q \in Q_1, q' \in Q_2} t \xrightarrow[\mathcal{A}_1]{*} q(t) \wedge t \xrightarrow[\mathcal{A}_2]{*} q'(t)}$ :

*In these cases the machine $\mathcal{A}$ will have a run $t \xrightarrow[\mathcal{A}]{*} (q, q')(t)$ in which the following cases can occur:*

$\underline{q \in Q_{1f} \wedge q' \in Q_{2f}}$ : *From Definition 4.14 follows that if both states $q$ and $q'$ are accepting states, the state $(q, q')$ is an accepting state. Therefore a tree $t$ is accepted by $\mathcal{A}$ if both $\mathcal{A}_2$ and $\mathcal{A}_2$ accept this tree.*

$\underline{q \notin Q_{1f} \vee q' \notin Q_{2f}}$ : *From Definition 4.14 follows that if either of the states $q$ and $q'$ is not an accepting state, the state $(q, q')$ will not be an accepting state. Therefore a tree $t$ is rejected by $\mathcal{A}$ if either $\mathcal{A}_2$ and/or $\mathcal{A}_2$ rejects this tree.*

*Therefore $\mathcal{A}$ accepts trees only if both $\mathcal{A}_1$ and $\mathcal{A}_2$ accept them, thus $L = \mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}_1) \cap \mathcal{L}(\mathcal{A}_2) = L_1 \cap L_2$.*

---

# 5 Top down Automata

Previous sections of this document have described *bottom-up automata*: Tree automata that start working in the leaves and work their way up. Similarly a second type of tree automaton exists that works the other way around: *Top down Automata*. These automata start working at the head and work their way down to the leaves. Section 1.7 of TATA[1] describes these automata briefly, yet we describe them in more detail.

## 5.1 Basics

For us to be able to use Top down tree automata we must first define them, their components and their rules.

**Definition 5.1.** *A top down tree automaton over $\mathcal{F}$ is a tuple $\mathcal{A} = (Q, \mathcal{F}, I, \Delta)$ with:*

- $Q$ : *the set of states, which are unary.*
- $\mathcal{F}$ : *the set of symbols.*
- $I$ : *the set of initial states, for which $I \subseteq Q$.*
- $\Delta$ : *the set of rewrite rules.*

**Definition 5.2.** *Rewrite rules are of the form:*

$$q(f(x_1, \ldots, x_n)) \rightarrow f(q_1(x_1), \ldots, q_n(x_n))$$

*where $n \geq 0, f \in \mathcal{F}_n, q, q_1, \ldots, q_n \in Q, x_1, \ldots, x_n \in V$. When $n = 0$ this means that the rewrite rule is of the form:*

$$q(a) \rightarrow a$$

*which occurs in the leaves of the tree.*

**Definition 5.3.** *Let $\mathcal{A} = (Q, \mathcal{F}, I, \Delta)$ be a Top down Tree Automaton over $\mathcal{F}$, The move relation $t \underset{\mathcal{A}}{\longrightarrow} t'$ for trees $t, t' \in T(\mathcal{F} \cup Q)$ is defined as:*

$$t \underset{\mathcal{A}}{\longrightarrow} t' := \begin{cases} \exists C \in K(\mathcal{F} \cup Q), \exists u_1, \ldots, u_n \in T(\mathcal{F})), \\ \exists q(f(x_1, \ldots, x_n)) \to f(q_1(x_1), \ldots, q_n(x_n)) \in \Delta, \\ t = C[q(f(u_1, \ldots, u_n))], \\ t' = C[f(q_1(u_1), \ldots, q_n(u_n))]. \end{cases} \tag{2}$$

*$\underset{\mathcal{A}}{\overset{*}{\longrightarrow}}$ is the transitive and reflexive closure of $\underset{\mathcal{A}}{\longrightarrow}$.*

**Definition 5.4.** *The language $L$ recognized by $\mathcal{A} = (Q, \mathcal{F}, I, \Delta)$ is the set of all ground terms $t$ such that:*

$$\exists_{q \in I}(q(t) \underset{\mathcal{A}}{\overset{*}{\longrightarrow}} t)$$

*A tree is rejected when for every initial state the tree gets stuck.*

## 5.2 Types of Top down automata

Now that we have given the basic definitions for the workings of Top down automata we define the types of Top down automata.

**Definition 5.5.** *A Top down tree automaton that adheres to the definitions and restrictions mentioned in Definitions 5.1, 5.2, 5.3 and 5.4 is called a **Top down NFTA**.*

---

**Example 5.6.** *Let us consider a Top down NFTA with $\mathcal{A} = (Q, \mathcal{F}, I, \Delta)$, where:*

$$Q = \{q_1, q_2, q_i, q_j, q_k\}$$

$$\mathcal{F} = \{a, f(), g(,)\}$$

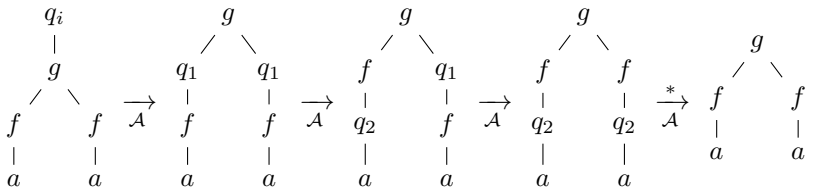
$$I = \{q_i, q_j, q_k\}$$

$$\Delta = \{q_2(a) \to a, \quad q_1(f(x)) \to f(q_2(x)), \\ q_i(g(x,y)) \to g(q_1(x), q_1(y)), \quad q_k(a) \to a\}$$

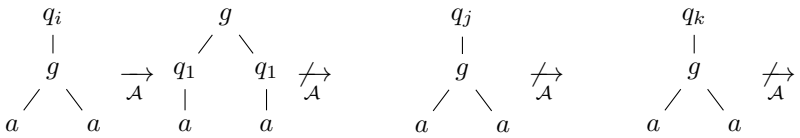
*and trees $t_1 = a, t_2 = g(f(a), f(a))$ and $t_3 = g(a, a)$. We will start by showing all possible runs on $t_1$:*

$$\begin{array}{ccc} q_k & & q_j \\ | & \underset{\mathcal{A}}{\longrightarrow} & | \\ a & & a \end{array} a \qquad \begin{array}{ccc} q_j & & \\ | & \not\underset{\mathcal{A}}{\longrightarrow} & \\ a & & \end{array} \qquad \begin{array}{ccc} q_i & & \\ | & \not\underset{\mathcal{A}}{\longrightarrow} & \\ a & & \end{array}$$

*As is shown, only the run starting with $q_k$ has any possible reductions. In that run the tree is accepted, in the runs starting with other initial states it gets stuck and is rejected. The machine however does have a run that accepts $t_1$, and therefore $\mathcal{A}$ accepts $t_1$. We will now show the successful run on $t_2$, not showing the ones that get stuck at start:*



*In which the last step the rule $q_2(a) \to a$ is applied to both branches, leading to the term $t_2$, thus accepting $t_2$. Now if we move on to $t_3$ there are no paths starting from in initial state and leading towards $t_3$ itself:*



*Therefore $t_3$ is not accepted. Considering the rules of $\mathcal{A}$ we can state that $\mathcal{L}(\mathcal{A}) = \{a, f(g(a))\}$*

---

**Definition 5.7.** *A **Top down DFTA** has several extra restrictions in than a Top down NFTA (5.5):*

- *$|I| = 1$, there is only one single initial state.*
- *No two rules in $\Delta$ share the same left hand side; there is at most one possible transition for every input.*

---

**Example 5.8.** *The automaton in Example 5.6 is not a DFTA, since it has multiple initial states.*

---

**Example 5.9.** *Consider the top down DFTA $\mathcal{A} = (Q, \mathcal{F}, I, \Delta)$ with:*

$$Q = \{q_1, q_i\}$$

$$\mathcal{F} = \{a, b, f()\}$$

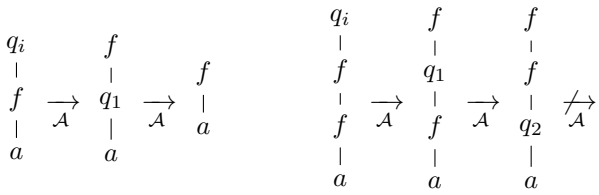
$$I = \{q_i\}$$

$$\Delta = \{q_i(f(x)) \to f(q_1(x)), \quad q_1(f(x)) \to (f(q_i(x))),$$
$$q_1(a) \to a, \quad q_i(b) \to b \quad \}$$

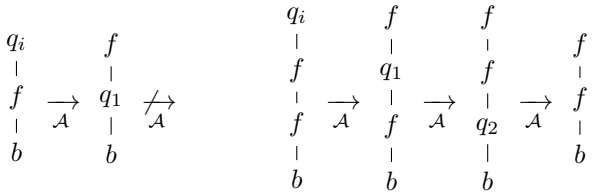
*$\mathcal{A}$ is a valid top down DFTA since $|I| = 1$ and no two rules in $\Delta$ share the same left hand rule.*

*Now let us look at the trees $t_1 = a, t_2 = b, t_3 = f(a), t_4 = f(f(a)), t_5 = f(b)$ and $t_6 = f(f(b))$. First we will show the runs on $t_1$ and $t_2$:*

*As we can see on both $t_1$ and $t_2$ the machine gets stuck, rejecting these trees. Now for $t_3$ and $t_4$*



*We can see that $t_3$ is accepted and $t_4$ rejected. Finally there are $t_5$ and $t_6$:*



*Where $t_5$ is rejected and $t_6$ is accepted. Considering the rules and initial states we can say that $\mathcal{L}(\mathcal{A}) = \{f^{2n+1}(a), f^{2n}(b) | n \geq 0\}$.*

## 5.3 Comparison of top down and bottom up automata

TATA[1] states that, because of the restrictions on a Top down DFTA, it is strictly less powerful than a Top down NFTA; and that Top down NFTA's are equally powerful as bottom up NFTA. TATA[1] does not provide the proof of this, but only some example exercises. This section will fill some of the missing information on the comparison of expressive power between these types of automata.

### 5.3.1 Top-down and bottom-up NFTA equivalence

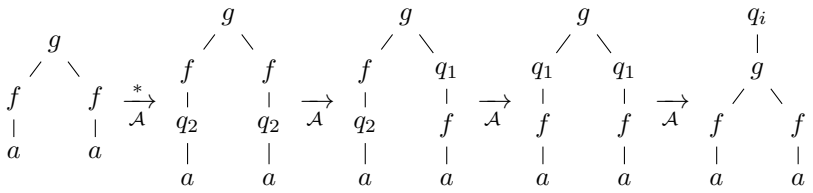
**Theorem 5.10.** *Top down NFTA and bottom up NFTA's accept the same class of languages and therefore have the same expressive power: For every top-down NFTA $\mathcal{A}_t$ there exists a bottom up NFTA $\mathcal{A}_b$ with $\mathcal{L}(\mathcal{A}_t) = \mathcal{L}(\mathcal{A}_b)$, and vice versa.*

In order to make things easier we create a definition to transform a set of transition rules for a bottom up NFTA into an equivalent set of rewrite rules for a top down NFTA.

**Definition 5.11.** *Given a set of rules $\Delta$, $\Delta^I$ is the set of inversed rules obtained by:*

$$\forall_{l \to r \in \Delta} r \to l$$

**Example 5.12.** *Given the Top down NFTA from Example 5.6, we can construct an equivalent machine* $\mathcal{A}_b = (Q, \mathcal{F}, Q_f, \Delta_b)$ *with* $Q_f = I$ *and* $\Delta_b = \Delta^I$*. For example a run on* $t_2$ *by* $\mathcal{A}_b$ *will look like:*



*Which is the reversed path of the run on* $t_2$ *by* $\mathcal{A}$ *in 5.6. Since now* $q_i \in Q_f$*, this tree is accepted by* $\mathcal{A}_b$*, which it also is by* $\mathcal{A}$*.*

---

**Proof of (5.10).** *Given a top-down NFTA* $\mathcal{A}_t = (Q, \mathcal{F}, I, \Delta)$ *and the bottom up NFTA* $\mathcal{A}_b = (Q, \mathcal{F}, Q_f, \Delta^I)$ *where* $Q_f = I$*. Given any rule*

$$q(f(x_1, \ldots, x_2)) \to f(q_1(x_1), \ldots, q_n(x_n)) \in \Delta,$$

*there exists a rule*

$$f(q_1(x_1), \ldots, q_n(x_n)) \to q(f(x_1, \ldots, x_2)) \in \Delta^I,$$

*therefore:*

$$q(t) \xrightarrow[\mathcal{A}_t]{*} t \iff t \xrightarrow[\mathcal{A}_b]{*} q(t)$$

*Now since* $Q_f = I$ *we get:*

$$q_i(t) \xrightarrow[\mathcal{A}_t]{*} t, q_i \in I \iff t \xrightarrow[\mathcal{A}_b]{*} q_f(t), q_f \in Q_f \text{ where } q_i = q_f$$

*Therefore* $\mathcal{L}(\mathcal{A}_b) = \mathcal{L}(\mathcal{A}_t)$

---

Since any bottom-up NFTA has a equivalent bottom up DFTA, and any NFTA-$\varepsilon$ has an equivalent bottom-up NFTA, the same can be said for top-down NFTA's having equivalent bottom-up DFTA's NFTA-$\varepsilon$'s.

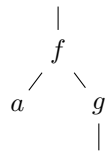
### 5.3.2 Top-down NFTA vs Top-down DFTA

Top-down DFTA's are strictly less powerful than top-down NFTA's: There are recognizable tree languages that cannot be accepted by a any top-down DFTA. We will prove this by giving two examples of recognizable tree languages that cannot be accepted by any top down DFTA.

**Example 5.13.** *Let us consider the following symbols and language as proposed in exercise 1.2[1]:* $\mathcal{F} = \{f(,), g(), a\}$ *and a language:*

$$L = \{C[f(a, g(u))] | C \in K(\mathcal{F}), u \in T(\mathcal{F})\}$$

*which are trees containing in their tree:*



*Let us consider the complete bottom up DFTA* $\mathcal{A} = (Q, \mathcal{F}, Q_f, \Delta)$ *with:*

$$Q = \{q_a, q_g, q, q_L\}$$

$$Q_f = \{q_L\}$$

$$\begin{aligned}
\Delta = \{ & a \to q_a(a), & & g(q_a(x)) \to q_g(g(x)), \\
& g(q_L(x)) \to q_L(g(x)), & & g(q(x)) \to q_g(g(x)), \\
& g(q_g(x)) \to q_g(g(x)), & & f(q(x), q(y)) \to q(f(x, y)), \\
& f(q_a(x), q_g(y)) \to q_L(f(x, y)), & & f(q_a(x), q_a(y)) \to q(f(x, y)), \\
& f(q_g(x), q_a(y)) \to q(f(x, y)), & & f(q_g(x), q_g(y)) \to q(f(x, y)), \\
& f(q(x), q_a(y)) \to q(f(x, y)), & & f(q_a(x), q(y)) \to q(f(x, y)), \\
& f(q(x), q_g(y)) \to q(f(x, y)), & & f(q_g(x), q(y)) \to q(f(x, y)), \\
& f(q_a(x), q_L(y)) \to q_L(f(x, y)), & & f(q_L(x), q_a(y)) \to q_L(f(x, y)), \\
& f(q_g(x), q_L(y)) \to q_L(f(x, y)), & & f(q_L(x), q_g(y)) \to q_L(f(x, y)), \\
& f(q(x), q_L(y)) \to q_L(f(x, y)), & & f(q_L(x), q(y)) \to q_L(f(x, y)), \\
& f(q_L(x), q_L(y)) \to q_L(f(x, y)) & & \}
\end{aligned}$$

*which goes into the accepting state* $q_L$ *when* $f(a, g(x))$ *is found and pushes this accepting state up to top of the tree. Therefore* $\mathcal{L}(\mathcal{A}) = L$

*Now we introduce the top-down NFTA* $\mathcal{A}_t = (Q, \mathcal{F}, I, \Delta_t)$*, with* $I = Q_f, \Delta_t = \Delta^I$*. According to Theorem 5.10* $\mathcal{L}(\mathcal{A}_t) = \mathcal{L}(\mathcal{A}) = L$*. We note that* $\mathcal{A}_t$ *has become nondeterministic by the inversion of rules.*

*It is impossible to construct any top-down DFTA that accepts L however. Using a single initial state would require a different DFTA for every possible variation of u. There are however an infinite number of possible variations of u, and therefore it would require a union of infinite top-down DFTA's to recognize L.*

---

**Example 5.14.** *Let* $\mathcal{F} = \{f(,), a, b\}$*. Consider the recognizable tree language* $L = \{f(a, b), f(b, a)\}$*. We will start by showing that L is recognizable. Let us consider an bottom-up automaton* $\mathcal{A} = (Q, \mathcal{F}, Q_f, \Delta)$ *with:*

$$Q = \{q_a, q_b, q_f\}$$

$$Q_f = \{q_f\}$$

$$\Delta = \{a \to q_a(a), \quad f(q_a(x), q_b(y)) \to q_f(f(x,y)),$$
$$b \to q_b(b), \quad f(q_b(x), q_a(y)) \to q_f(f(x,y)) \quad \}$$

*$\mathcal{A}$ recognizes L, and so does its Top down equivalent $\mathcal{A}_t = (Q, \mathcal{F}, I, \Delta^I)$ in which $I = Q_f$.*

*Let us consider the top-down DFTA $\mathcal{A}_d$ that recognizes L. We know it has only one initial state, $q_i \in I$ (see Definition 5.7), and it must be able to remove states from the leaves of it's input tree, leading to some rules $q(a) \to a$ and $q'(b) \to b$ for some states $q$ and $q'$.*

*Now if $q \neq q'$ we will require two rules $q^1(f(x,y)) \to f(q(x), q'(y))$ and $q^2(f(x,y)) \to f(q'(x), q(y))$ for some $q^1, q^2$. If we pick $q^1 = q^2$ the machine would no longer be a DFTA, and if we pick $q^1 \neq q^2$ we can only pick one of either states to be $q_i$, therefore only accepting part of L.*

*Now if $q = q'$ we will require a rule q"$(f(x,y)) \to f(q'(x), q'(y))$. As a result of this choice, both $f(a,a)$ and $f(b,b)$ would also be accepted, therefore $\mathcal{L}(\mathcal{A}_d) \neq L$*

---

### 5.3.3 Path closure

Top down DFTA's do not accept all **regular tree languages**(The class of languages which are recognized by NFTA's). They do, however, recognize another class of tree languages known as **path-closed tree languages**. TATA[1] briefly adresses these in exercise 1.8, which will be worked out in this section.

**Definition 5.15.** *Let $t$ be a ground term , the path language $\pi(t)$ is then inductively defined by:*

- *if $t \in \mathcal{F}_0$ then $\pi(t) = t$*
- *if $t = f(t_1, \ldots, t_n)$ then $\pi(t) = \bigcup_{i=1}^{i=n} \{fiw | w \in \pi(t_i)\}$*

*Let L be a tree language, the path language of L is defined as $\bigcup_{t \in L} \pi(t)$, the path closure of L is defined as $pathclosure(L) = \{t | \pi(t) \subseteq \pi(L)\}$. A tree language L is path closed if $pathclosure(L) = L$.*

---

**Example 5.16.** *Given $\mathcal{F} = \{f(), g(,), a\}$ and trees $t_1 = a$, $t_2 = f(f(a))$ and $t_3 = g(f(a), a)$ we have:*

$$\pi(t_1) = \{a\}$$

$$\pi(t_2) = \{f1f1a\}$$

$$\pi(t_3) = \{g1f1a, g2a\}$$

*Consider* $L = \{t_1, t_2, t_3\}$. *we then get:*

$\pi(L) = \{a, f1f1a, g1f1a, g2a\}$

$pathclosure(L) = \{t_1, t_2, t_3\}$

$pathclosure(L) = L$ *thus* $L$ *is path closed.*

---

**Theorem 5.17.** *A language* $L$ *is accepted by a top down DFTA if and only if* $L$ *is path closed.*

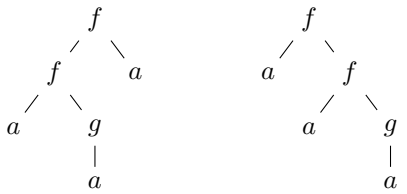
We will not prove this theorem, yet we will use this theorem to prove that the languages in Examples 5.13 and 5.14 are not accepted by any top down DFTA's.

---

**Lemma 5.18.** *The language* $L$ *of Exercise 5.13 can not be accepted by a top down DFTA.*

---

**Proof of** (5.18)**.** *Given the language in Example 5.13, we will consider the following subtrees* $t_1 = f(f(a, g(a)), a)$ *and* $t_2 = f(a, f(a, g(a)))$*:*




*which are both in* $L$*. We then get*

$\pi(t_1) = \{f1f1a, f1f2g1a, f2a\}$

$\pi(t_2) = \{f1a, f2f1a, f2f2g1a\}$

$\pi(t_1) \subset \pi(L)$ *and* $\pi(t_2) \subset \pi(L)$

*Now if we look at* $\pi(L)$*, due to* $f2a$ *existing in* $\pi(t_1)$ *and* $f1a$ *existing in* $\pi(t_2)$*,* $f1a$ *and* $f2a$ *are both in* $\pi(L)$*.*

*Therefore* $f(a, a) \in pathclosure(L)$*, while* $f(a, a) \notin L$*, thus* $L \neq pathclosure(L)$*, which means that* $L$ *is not path closed. Therefore, according to Theorem 5.17,* $L$ *cannot be accepted by a top down DFTA.*

---

**Lemma 5.19.** *The language* $L$ *of Exercise 5.14 can not be accepted by a top down DFTA.*

---

**Proof of** (5.19)**.** *Given the language in Example 5.14, we get*

$\pi(L) = \{f1a, f1b, f2a, f2b\}$

$pathclosure(L) = \{f(a,a), f(a,b), f(b,a), f(b,b)\}$

$pathclosure(L) \neq L$ *thus* $L$ *is not path closed.*

*In this language, the path closure of* $L$ *is a larger set than* $L$ *itself. Therefore, according to Theorem 5.17,* $L$ *cannot be accepted by a top down DFTA.*

---

**Theorem 5.20.** *Top down DFTA's are strictly less powerful than top down NFTA's. There are recognizable tree languages that cannot be accepted by any top down DFTA.*

---

**Proof of** (5.20)**.** *Lemma's 5.18 and 5.19 show two regular tree languages that can not be accepted by any top down DFTA.*

# 6 Conclusion

In this document we have described tree automata and their properties. We have described the different forms of tree automata and their different properties, provided algorithms for transforming various types of tree automata into different ones and added examples and proofs where needed.

This document was created to clarify the contents of the first chapter of TATA[1], for which numerous examples have been added. We have also added algorithms for DFTA completion (algorithm 3.20) and epsilon removal (algorithm 3.35), provided the proofs for the Language properties for tree automata, and expanded on the Top down automata section in TATA[1].

# References


[1] H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Available on: http://www.grappa.univ-lille3.fr/tata, 2007. release October, 12th 2007.

[2] J. E. Doner. Decidability of the weak second-order theory of two successors. Notices Amer. Math. Soc., 12: 365 – 468, 1965.

[3] J. E. Doner. Tree acceptors and some of their applications. Journal of Comput. and Syst. Sci, 4:406 – 451, 1970.

[4] J. E. Hopcroft and J. D. Ullman. Introduction to automata theory, languages, and computation. Addison Wesley, 1979.

[5] J. W. Thatcher and J. B. Wright. Generalized finite automata. Notices Amer. Math. Soc., 820, 1965. Abstract No 65T–649, 1965.

[6] J. W. Thatcher and J. B. Wright. Generalized finite automata with an application to a decision problem of second-order logic. Mathematical System Theory, 2:57-82, 1968.