# Chapter 1

# Tree Acceptors

## 1.1 Deterministic Bottom-up Finite-state Tree Acceptors

### 1.1.1 Orientation

This section is about deterministic bottom-up finite-state tree acceptors. The term *finite-state* means that the memory is bounded by a constant, no matter the size of the input to the machine. The term *deterministic* means there is single course of action the machine follows to compute its output. The term *acceptor* means this machine solves *membership problem*: given a set of objects $X$ and input object $x$, does $x$ belong to $X$? The term *tree* means we are considering the membership problem over *treesets*. The term *bottom-up* means that for each node $a$ in a tree, the computation solves the problem by assigning states to the children of $a$ before assigning states to $a$ itself. This contrasts with *top-down* machines which assign states to $a$ first and then the children of $a$. Visually, these terms make sense provided the root of the tree is at the top and branches of the tree move downward.

*Acceptor* is synonymous with *recognizer*. *Treeset* is synonymous with *tree language*.

A definitive reference for finite-state automata for trees is freely available online. It is "Tree Automata Techniques and Applications" (TATA) (Comon *et al.*, 2007). The presentation here differs from the one there, as mentioned below.

### 1.1.2 Definitions

We will use the following definition of trees.

**Definition 1** (Trees). *We assume an alphabet* $\Sigma$ *and symbols [ ] not belonging to* $\Sigma$.

**Base Cases:** *For each* $a \in \Sigma$, $a[\,]$ *is a tree.*

**Inductive Case:** *If* $a \in \Sigma$ *and* $t_1 t_2 \dots t_n$ *is a string of trees of length* $n$ *then* $a[t_1 t_2 \cdot \dots t_n]$ *is a tree.*

*Let $\Sigma^T$ denote the set of all trees of finite size using $\Sigma$. We also write $a[\lambda]$ for $a[\,]$.*

It will be helpful to review the following concepts related to functions: *domain, co-domain, image*, and *pre-image*. A function $f : X \rightarrow Y$ is said to have domain $X$ and co-domain $Y$. This means that if $f(x)$ is defined, we know $x \in X$ and $f(x) \in Y$. However, $f$ may not be defined for all $x \in X$. Also, $f$ may not be *onto* $Y$, there may be some elements in $Y$ that are never "reached" by $f$.

This is where the concepts *image* and *pre-image* come into play. The image of $f$ is the set $\{f(x) \in Y \mid x \in X, f(x) \text{ is defined}\}$. The pre-image of $f$ is the set $\{x \in X \mid f(x) \text{ is defined}\}$. So the pre-image of $f$ is the subset of the domain of $f$ where $f$ is defined. The image of $f$ is the corresponding subset of the co-domain of $f$.

With this in place, we can define our first tree acceptor.

**Definition 2** (DBFTA)**.** *A* Deterministic Bottom-up Finite-state Acceptor (DBFTA) *is a tuple $(Q, \Sigma_r, F, \delta)$ where*

- *$Q$ is a finite set of states;*
- *$\Sigma$ is a finite alphabet;*
- *$F \subseteq Q$ is a set of* accepting *(*final*) states; and*
- *$\delta : Q^* \times \Sigma \rightarrow Q$ is* the transition function. The pre-image of $\delta$ must be finite. *This means we can write it down—for example, as a list.*

We use the transition function $\delta$ to define a new function $\delta^* : \Sigma^T \rightarrow Q$ as follows.

$$\begin{aligned} \delta^*(a[\lambda]) &= \delta(\lambda, a) \\ \delta^*(a[t_1 \cdots t_n]) &= \delta\big(\delta^*(t_1) \cdots \delta^*(t_n), a\big) \end{aligned} \tag{1.1}$$

There are some important consequences to the formulation of $\delta^*$ shown here. One is that $\delta^*$ is undefined on tree $a[t_1 \cdots t_n]$ if no transition $\delta(q_1 \cdots q_n, a)$ is defined.

(Also, I am abusing notation since $\delta^*$ is strictly speaking not the transitive closure of $\delta$.)

**Definition 3** (Treeset of a DBFTA)**.** *Consider some DBFTA $A = (Q, \Sigma, F, \delta)$ and tree $t \in \Sigma^T$. If $\delta^*(t)$ is defined and belongs to $F$ then we say $A$* accepts/recognizes *$t$. Otherwise $A$* rejects *$t$. The treeset recognized by $A$ is $L(A) = \{t \in \Sigma^T \mid \delta^*(t) \in F\}$.*

The use of the 'L' denotes "Language" as treesets are traditionally referred to as *formal tree languages.*

**Definition 4** (Recognizable Treesets)**.** *A treeset is* recognizable *if there is a DBFTA that recognizes it.*

### 1.1.3 Notes on Definitions

We have departed a bit from standard definitions. In particular, most introductions to tree automata make use of a particular kind of alphabet called a *ranked alphabet*. A ranked alphabet $\Sigma_r$ is an alphabet $\Sigma$ with an arity function $ar : \Sigma \to \mathbb{N}$. We write $\Sigma_r = (\Sigma, ar)$. The idea is that each symbol comes pre-equipped with a number which indicates how many children it has in trees. This is reasonable provided a node's label determines how many children it can have.

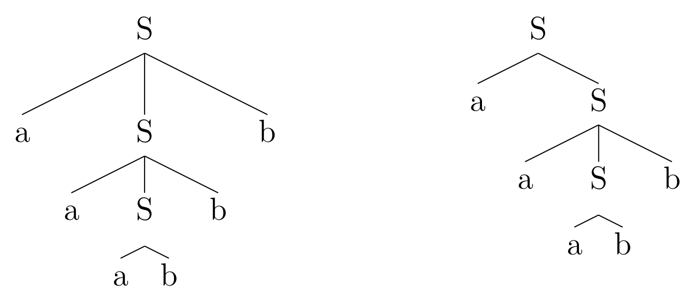
Strictly speaking, a ranked alphabet is not a necessary feature of tree automata. There are two substantive reasons to adopt it. First, using it helps ensure that the transition function is finite. (So it can accomplish the same thing as our requirement that the pre-image of $\delta$ be finite.). Second, it helps ensure our transition function is total; that is, defined for every element of the alphabet and the possible states of its children.

### 1.1.4 Examples

**Example 1.** Let $A = (Q, \Sigma, F, \delta)$ with its parts defined as follows.

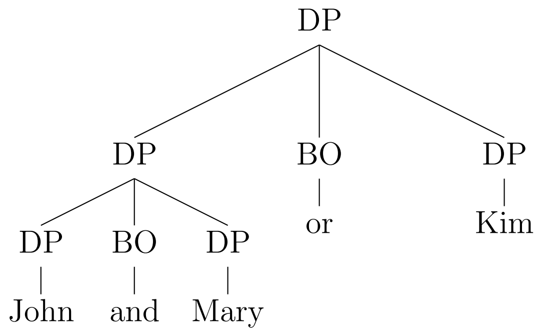
- $Q = \{q_a, q_b, q_S\}$
- $\Sigma = \{a, b, S\}$
- $F = \{q_S\}$
- $\delta(\lambda, a) = q_a$
- $\delta(\lambda, b) = q_b$
- $\delta(q_a q_b, S) = q_S$
- $\delta(q_a q_S q_b, S) = q_S$

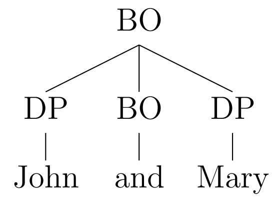
Let us see how the acceptor $A$ processes the two trees below as inputs.




**Example 2.** Let $A = (Q, \Sigma, F, \delta)$ with its parts defined as follows.

- $Q = \{q_{DP}, q_{BO}\}$
- $\Sigma = \{\text{and, or, Kim, John, Mary, DP, BO}\}$
- $F = \{q_{DP}\}$
- $\delta(q_{DP}, DP) = q_{DP}$
- $\delta(q_{BO}, BO) = q_{BO}$
- $\delta(q_{DP}\ q_{BO}\ q_{DP}, \text{DP}) = q_{DP}$
- $\delta(\lambda, \text{Kim}) = q_{DP}$
- $\delta(\lambda, \text{John}) = q_{DP}$
- $\delta(\lambda, \text{Mary}) = q_{DP}$
- $\delta(\lambda, \text{and}) = q_{BO}$
- $\delta(\lambda, \text{or}) = q_{BO}$




### 1.1.5 Observations

- For every symbol $a \in \Sigma$ which can be leaf in a tree, you will need to define a transition $\delta(\lambda, a)$.

- For every symbol $a \in \Sigma$ which can have $n$ children, you will need to define a transition $\delta(q_1 \cdots q_n, a)$.

### 1.1.6 Connection to Context-Free Languages

Recognizable treesets are closely related to the derivation trees of context-free languages.

**Theorem 1.**

- *Let $G$ be a context-free word grammar, then the set of derivation trees of $L(G)$ is a recognizable tree language.*
- *Let $L$ be a recognizable tree language then Yield(L) is a context-free word language.*
- *There exists a recognizable tree language not equal to the set of derivation trees of any context-free language. Thus the class of derivation treesets of context-free word languages is a proper subset of the class of recognizable treesets.*

## 1.2 Deterministic Top-down Finite-state Tree Acceptors

### 1.2.1 Orientation

This section is about deterministic top-down finite-state tree acceptors. The term *finite-state* means that the memory is bounded by a constant, no matter the size of the input to the machine. The term *deterministic* means there is single course of action the machine follows to compute its output. The term *acceptor* means this machine solves *membership problem*: given a set of objects $X$ and input object $x$, does $x$ belong to $X$? The term *tree* means we are considering the membership problem over *treesets*. The term *top-down* means that for each node $a$ in a tree, the computation solves the problem by assigning a state to the parent of $a$ before assigning a state to $a$ itself. This contrasts with *bottom-up* machines which assign states to the children of $a$ first and then $a$. Visually, these terms make sense provided the root of the tree is at the top and branches of the tree move downward.

*Acceptor* is synonymous with *recognizer*. *Treeset* is synonymous with *tree language*.

A definitive reference for finite-state automata for trees is freely available online. It is "Tree Automata Techniques and Applications" (TATA) (Comon *et al.*, 2007). The presentation here differs from the one there, as mentioned below.

### 1.2.2 Definition

**Definition 5** (DTFTA). *A* Deterministic Top-down Finite-state Acceptor (DTFTA) *is a tuple* $(Q, \Sigma_r, F, \delta)$ *where*

- $Q$ *is a finite set of states;*
- $q_0$ *is a initial state;*
- $\Sigma$ *is a finite alphabet;*
- $\delta : Q \times \Sigma \times \mathbb{N} \to Q^*$ *is the transition function. Note the pre-image of* $\delta$ *is necessarily finite.*

The transition function takes a state, a letter, and a number $n$ and returns a string of states. The idea is that the length of this output string should be $n$. Basically, when moving top-down, the states of the child sub-trees depend on these three things: the state of the parent, the label of the parent, and the number of children the parent has.

We use the transition function $\delta$ to define a new function $\delta^* : Q \times \Sigma^T \to Q^*$ as follows.

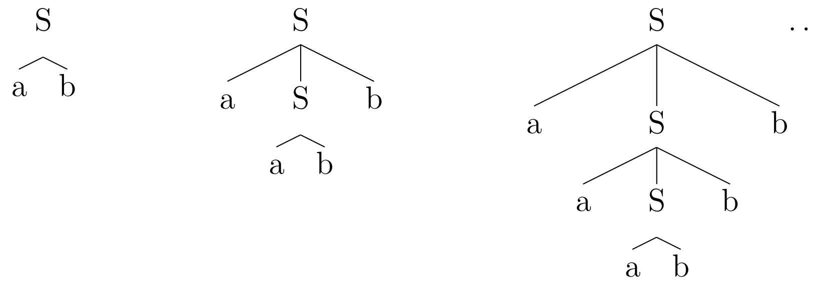
$$\begin{aligned} \delta^*(q, a[\lambda]) &= \delta(q, a, 0) \\ \delta^*(q, a[t_1 \cdots t_n]) &= \delta^*(q_1, t_1) \cdots \delta^*(q_n, t_n) \text{ where } \delta(q, a, n) = q_1 \cdots q_n \end{aligned} \tag{1.2}$$

As before, there are some important consequences to the formulation of $\delta^*$. One is that $\delta^*$ is undefined on tree $a[t_1 \cdots t_n]$ if transition $\delta(q, a, n)$ does not return a string from $Q^*$ of length $n$. (Also, I am abusing notation since $\delta^*$ is strictly speaking not the transitive closure of $\delta$.)

**Definition 6** (Treeset of a DTFTA). *Consider some DTFTA* $A = (Q, \Sigma, F, \delta)$ *and tree* $t \in \Sigma^T$. *If* $\delta^*(q_0, t)$ *is defined and equals* $\lambda$ *then we say* $A$ accepts/recognizes $t$. *Otherwise* $A$ rejects $t$. *Formally, the treeset recognized by* $A$ *is* $L(A) = \{t \in \Sigma^T \mid \delta^*(t) = \lambda\}$.

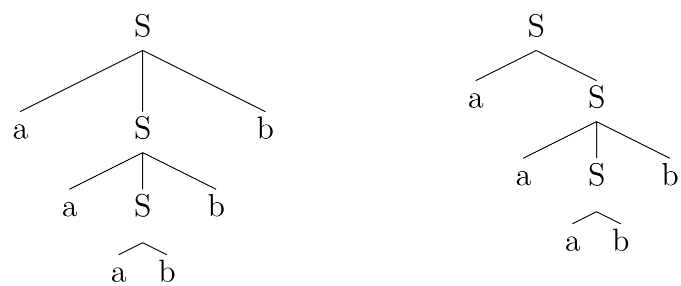
The use of the 'L' denotes "Language" as treesets are traditionally referred to as *formal tree languages*.

**Example 3.** Recall the example from last week which generates trees like



- $Q = \{q_a, q_b, q_S\}$
- $\Sigma = \{a, b, S\}$
- $q_0 = q_S$
- $\delta(q_a, a, 0) = \lambda$
- $\delta(q_b, b, 0) = \lambda$
- $\delta(q_S, S, 3) = q_a q_S q_b$
- $\delta(q_S, S, 2) = q_a q_b$

Let us see how the acceptor $A$ processes the two trees below as inputs.



**Theorem 2.** *Every treeset recognizable by a DTFTA is recognizable, but there are recognizable treesets which cannot be recognized by a DTFTA.*

The following example helps show why this is the case. Consider the treeset $T$ containing only the two trees shown below.

This is a recoginiable treeset because the DBFTA below recognizes exactly these two trees and no others.

- $Q = \{q, q_S\}$
- $\Sigma = \{a, b, S\}$
- $q_0 = q_S$
- $\delta(\lambda, a) = q_a$
- $\delta(\lambda, b) = q_b$
- $\delta(q_a q_b, S) = q_S$
- $\delta(q_b q_a, S) = q_S$

Notice that this DBFTA fails on these two trees.

A DTFTA cannot recognize the trees in $T$ without also recognizing the trees shown immediately above. This is because moving top down there can only be one value for $\delta(q_S, S, 2)$. Suppose it equals $q_1q_2$. To recognize the first tree, we would also have to makes sure that $\delta(q_1, a, 0)$ and $\delta(q_2, b, 0)$ are defined. Similarly, to recognize the second tree, we would have to makes sure that $\delta(q_1, b, 0)$ and $\delta(q_2, a, 0)$ are defined. But it follows then that the aforementioned trees above are also recognized by this DTFTA. For instance the tree with two $a$ leaves is recognized because both $\delta(q_1, a, 0)$ and $\delta(q_2, a, 0)$ are defined. Thus no DTFTA recognizes $T$.

### 1.2.3 Observations

- For every symbol $a \in \Sigma$ which can be leaf in a tree, you will need to define a transition $\delta(q, a, 0) = \lambda$.
- For every symbol $a \in \Sigma$ which can have $n$ children, you will need to define a transition $\delta(q, a, n) = q_1 \cdots q_n$.

## 1.3 Properties of recognizable tree languages

**Theorem 3** (Closure under Boolean operations)**.** *The class of recognizable tree languages is closed under union, under complementation, and under intersection.*

The proofs of these cases are very similar to the ones for finite-state acceptors over strings. For every DBFTA $A$ recognizing a treeset $T$, it can be made *complete* by adding a sink state and transitions to it. Then product constructions can be used to establish closure under union and intersection. Closure under complement is established the same as before too: everything is the same except the final states are now the non-final states of $A$.

**Theorem 4** (Minimal, determinstic, canonical form)**.** *For every recognizable tree language $T$, there is a unique, smallest DBFTA $A$ which recognizes $T$. That is, if DBFTA $A'$ also recognizes $T$ then there at least as many states in $A'$ as there are in $A$.*

## 1.4 Connection to Context-Free Languages

Context Free Grammars (CFGs) are studied in detail in a number of textbooks including Harrison (1978); Davis and Weyuker (1983); Hopcroft *et al.* (2001) and Sipser (1997).

**Definition 7.** *A* rewrite grammar *is a tuple $\langle T, N, S, \mathcal{R}\rangle$ where*

- $\mathcal{T}$ *is a nonempty finite alphabet of symbols. These symbols are also called the* terminal *symbols, and we usually write them with lowercase letters like $a, b, c, \ldots$*
- $\mathcal{N}$ *is a nonempty finite set of* non-terminal *symbols, which are distinct from elements of $\mathcal{T}$. These symbols are also called* category *symbols, and we usually write them with uppercase letters like $A, B, C, \ldots$*

- *$S$ is the* start *category, which is an element of $\mathcal{N}$.*
- *A finite set of* production rules *$\mathcal{R}$. A production rule has the form $\alpha \to \beta$ where $\alpha, \beta$ belong to $(\mathcal{T} \cup \mathcal{N})^*$. In other words, $\alpha$ and $\beta$ are strings of non-terminal and terminal symbols. While $\beta$ may be the empty string we require that $\alpha$ include at least one symbol.*

Rewrite grammars are also called *phrase structure grammars*.

**Definition 8.** *A* CFG *is a rewrite grammar with the following properties.*

- *For all rules $\alpha \to \beta$, $\alpha$ is an element of $\mathcal{N}$. So the left-hand-side of each rule is a single non-terminal.*
- *For all rules, $\alpha \to \beta$, beta is an element of $(\mathcal{N} \cup \mathcal{T})^*$. So the right-hand-side of each rule is a sequence of non-terminal symbols or a single terminal symbol.*

**Example 4.** Define a CFG $G_{NE}$ as follows. Let $\mathcal{N} = \{A, B, S\}$ and $\mathcal{T} = \{a, b\}$. Let $\mathcal{R}$ include the rules $S \to a\ S\ b$ and $S \to a\ b$.

Next we define the languages of the rewrite grammars in addition to the derivation treeset of context free grammars.

The language of a rewrite grammar is defined recursively below.

**Definition 9.** *The (partial)* derivations *of a rewrite grammar $G = \langle \mathcal{T}, \mathcal{N}, S, \mathcal{R} \rangle$ is written $D(G)$ and is defined recursively as follows.*

1. The base case: *$S$ belongs to $D(G)$.*
2. The recursive case: *For all $\alpha \to \beta \in \mathcal{R}$ and for all $\gamma_1, \gamma_2 \in (\mathcal{T} \cup \mathcal{N})^*$, if $\gamma_1 \alpha \gamma_2 \in D(G)$ then $\gamma_1 \beta \gamma_2 \in D(G)$.*
3. *Nothing else is in $D(G)$.*

*Then the language of the grammar $L(G) = \big\{ w \in \mathcal{T}^* \mid w \in D(G) \big\}$.*

**Exercise 1.** Using the definition above, explain why *aaabbb* belongs to $L(G_{NE})$.

The derivation treeset of a context free grammar is defined recursively below. The *derivation treeset* of a CFG $G = \langle \mathcal{T}, \mathcal{N}, S, \mathcal{R} \rangle$ is written $D_T(G)$.

It is defined as all and only those trees $t$ such that

1. $\texttt{yield}(t) \in L(G)$ and
2. for all non-leaf nodes $a[a_1[ts_1]a_2[ts_2] \cdots a_n[ts_n]]$ in $t$, the rule $a \to a_1 a_2 \cdots a_n$ belongs to $\mathcal{R}$.

Recognizable treesets are closely related to the derivation trees of context-free languages.

**Theorem 5.**

- *Let $G$ be a context-free word grammar, then the set of derivation trees $D_T(G)$ is a recognizable tree language.*

- *There exists a recognizable tree language not equal to the set of derivation trees of any context-free language. Thus the class of derivation treesets of context-free word languages is a proper subset of the class of recognizable treesets.*
- *Let $L$ be a recognizable tree language then* $\texttt{yield}(L)$ *is a context-free word language.*

Each of these has a straightforward explanation.

For (1), the recognizable tree language which recognizes $D_T(G)$ for a CFG $G$ can be constructed based on the rules of $G$. For each symbol $a$ in $N$, the DBFTA should include $\delta(\lambda, a) = q_a$. And, for each rule $A \to B_1 \cdots B_n$ in $R$, the DBFTA should include $\delta(q_{B_1} \cdots q_{B_n}, A) = q_A$. That's it!

For (2), consider the DBFTA $A$ shown below. The claim is that there is no CFG whose derivation language is exactly this recognizable treeset. The only tree in $L(A)$ is (t1), which is shown below $A$ at left. Tree (t2) shows (t1) with the states $A$ assigns to its subtrees.

$$(A) \qquad \begin{array}{rcl} Q &=& \{q_S, q_a, q_b, q_x, q_y\} \\ \Sigma &=& \{S, G, a, b\} \\ F &=& \{q_S\} \\ \delta(q_x q_y, S) &=& q_S \end{array} \qquad \begin{array}{rcl} \delta(\lambda, a) &=& q_a \\ \delta(\lambda, b) &=& q_b \\ \delta(q_a, G) &=& q_x \\ \delta(q_b, G) &=& q_y \end{array}$$

(t1) [S [G a] [G b]]

(t2) [S ($q_S$) [G ($q_x$) a ($q_a$)] [G ($q_y$) b ($q_b$)]]

That no CFG can recognize this treeset follows from the fact that such any CFG $G$ which includes the tree above will need to have the following rules: $S \to GG, G \to a, G \to b$. But then this $G$ will not only generate (t1) above but also the derivation trees shown below.

(t3) [S [G a] [G a]]

(t4) [S [G b] [G a]]

(t5) [S [G b] [G b]]

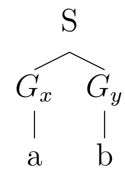
Thus $L(A) \neq D_T(G)$.

This example shows that the states of the DBFTA are more abstract than the labels on the nodes. The reason recognizable tree languages are more expressive than the derivation treesets of CFGs follows from this. The DBFTA uses states $q_x$ and $q_y$ to distinguish the subtrees bearing the label $G$. But the CFG cannot distinguish these trees in this way.

For (3), observe that we can write a CFG that generates the same stringset as the one above. For instance, we could write a CFG with the rule $S \to ab$.

More generally, though, we can always write a CFG that puts the state information into the nodes themselves. The trees in the derivation treeset for this CFG would be "structurally the same" as the trees in the recognizable treeset, but the labels on the nodes would be different. So they are not the same trees. In the example above for instance we can write a CFG with rules $S \to G_x G_y, G_x \to a, G_x \to b$. There is only one tree in this CFG's derivation treeset shown below.

(t6)




Importantly, (t6) is not the same as (t1). The inner nodes are labeled differently! So they are different trees. However, they only differ with respect to how the inner nodes are labeled, so it follows that the stringsets obtained by taking the `yield` of these trees are the same. This is the kind of argument used to show that the yield of any recognizable treeset is a context-free language.

# Chapter 2

# Tree Transducers

## 2.1 Deterministic Bottom-up Finite-state Tree Transducers

### 2.1.1 Orientation

This section is about deterministic bottom-up finite-state tree transducers. The term *finite-state* means that the amount of memory needed in the course of computation is independent of the size of the input. The term *deterministic* means there is single course of action the machine follows to compute its output. The term *transducer* means this machine solves *transformation problem*: given an input object $x$, what object $y$ is $x$ transformed into? The term *tree* means we are considering the transformation problem from trees to trees. The term *bottom-up* means that for each node $a$ in a tree, the computation transforms the children of a node before transforming the node. This contrasts with *top-down* transducers which transform the children after transforming their parent. Visually, these terms make sense provided the root of the tree is at the top and branches of the tree move downward.

A definitive reference for finite-state automata for trees is freely available online. It is "Tree Automata Techniques and Applications" (TATA) (Comon *et al.*, 2007). The presentation here differs from the one there, as mentionepd below.

### 2.1.2 Definitions

Recall the definition of trees with a finite alphabet $\Sigma$ and the symbols [ ] not in $\Sigma$. All such trees belonged to the treeset $\Sigma^T$. In addition to this, we will need to define a new kind of tree which has *variables* in the leaves. I will call these trees *Variably-Leafed*. We assume a countable set of variables $X$ containing variables $x_1, x_2, \ldots$.

**Definition 10** (Variably-Leafed Trees)**.**

**Base Cases ($\Sigma$):** *For each $a \in \Sigma$, $a[\,]$ is a tree.*
**Base Cases ($X$):** *For each $x \in X$, $x[\,]$ is a tree.*

**Inductive Case:** *If $a \in \Sigma$ and $t_1 t_2 \dots t_n$ is a string of trees of length $n$ then $a[t_1 t_2 \cdot \dots t_n]$ is a tree.*

*Let $\Sigma^T[X]$ denote the set of all variably-leafed trees of finite size using $\Sigma$ and $X$.*

Note that $\Sigma^T \subsetneq \Sigma^T[X]$. In the tree transducers we write below, the variably-leafed trees will play a role in the omega function as well as the the intermediate stages of the transformation.

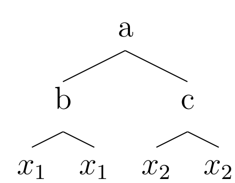
With this definition in place, we can define our first tree transducer.

**Definition 11** (DBFTA). *A* Deterministic Bottom-up Finite-state Acceptor (DBFTT) *is a tuple $(Q, \Sigma_r, F, \delta)$ where*

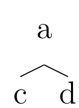
- *$Q$ is a finite set of states;*
- *$\Sigma$ is a finite alphabet;*
- *$F \subseteq Q$ is a set of* accepting *(*final*) states; and*
- *$\delta : Q^* \times \Sigma \to Q$ is* the transition function. The pre-image of $\delta$ must be finite. *This means we can write it down—for example, as a list.*
- *$\Omega$ is a function with domain $Q^* \times \Sigma$ and co-domain $\Sigma^T[X]$.* Its pre-image must also be finite.

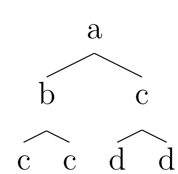
*Generally, the pre-images of $\delta$ and $\Omega$ should coincide.*

**Example 5.** Let $M$ be a DBFTT and suppose $\Omega(q_1, q_2, a) = a\big[\ b[x_1\ x_1]\ \ c[x_2\ x_2]\ \big]$. So this is the variably-leafed tree shown below.



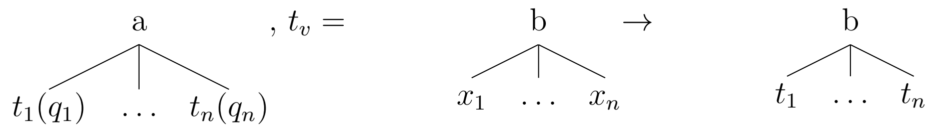
Further suppose $\Omega(\lambda, c) = c[\ ]$ and $\Omega(\lambda, d) = d[\ ]$. The idea is that $M$ will transform the tree shown below at left into the tree shown below at right.



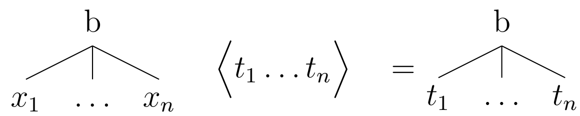


Intuitively, this is because each variable $x_i$ in the variably leafed tree will be replaced by the $i$th child of the root node of the tree under consideration.

Here is the general schema. Given $\Omega(q_1 q_2 \dots q_n, a) = t_v$ and a tree $a[t_1 t_2 \dots t_n]$ with states $q_1 q_2 \dots q_n$, respectively, then the output tree will be the one obtained by replacing each $x_i$ with $t_i$ in $t_v$. A schematic of this is shown below.




We now formalize the above ideas. As before, we extend the transition function $\delta$ to $\delta^* : \Sigma^T \to Q$. If $t_1 \dots t_n$ is a list of trees and $t_v \in \Sigma^T[X]$ is a variably leafed tree with variables $x_1, \dots x_n$ then let $t_v\langle t_1 \dots t_n\rangle = t \in \Sigma^T$ obtained by replacing each variable $x_i$ in $t_v$ with $t_i$. Here is another visualization with this notation.




We also define a new function "process" $\pi : \Sigma^T \to Q \times \Sigma^T$ which will process the tree and produce its output. It is defined as follows.

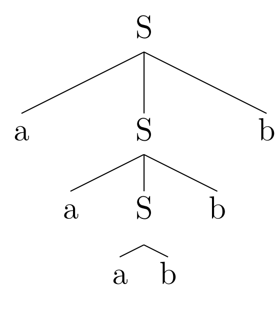
$$\begin{aligned}
\pi(a[\;]) &= \big(\delta(\lambda, a), \Omega(\lambda, a)\big) \\
\pi(a[t_1 \cdots t_n]) &= (q, t) \\
& \quad \text{where } q = \delta\big(q_1 \cdots q_n, a\big) \\
& \quad \text{and } t = \Omega\big(q_1 \cdots q_n, a\big)\langle s_1 \cdots s_n\rangle \\
& \quad \text{and } (q_1, s_1) \cdots (q_n, s_n) = \pi(t_1) \cdots \pi(t_n)
\end{aligned} \tag{2.1}$$

**Definition 12** (Tree-to-tree function of a DBFTT)**.** *The function defined by the transducer $M$ is $\big\{(t,s) \mid t, s \in \Sigma^T, \pi(t) = (q,s), q \in F\big\}$. If $(t,s)$ belongs to this set, we say $M$* transduces *$t$ to $s$ and write $M(t) = s$.*

**Example 6.** Consider the transducer $M$ constructed as follows.

- $Q = \{q_a, q_b, q_S\}$
- $\Sigma = \{a, b, S\}$
- $F = \{q_S\}$
- $\delta(\lambda, a) = q_a$
- $\delta(\lambda, b) = q_b$
- $\delta(q_a q_b, S) = q_S$
- $\delta(q_a q_S q_b, S) = q_S$
- $\Omega(\lambda, a) = a[\;]$
- $\Omega(\lambda, b) = b[\;]$
- $\Omega(q_a q_b, S) = S[x_2 x_1]$
- $\Omega(q_a q_S q_b, S) = S[x_3 x_2 x_1]$

Let us work out what $M$ transforms the tree below into.

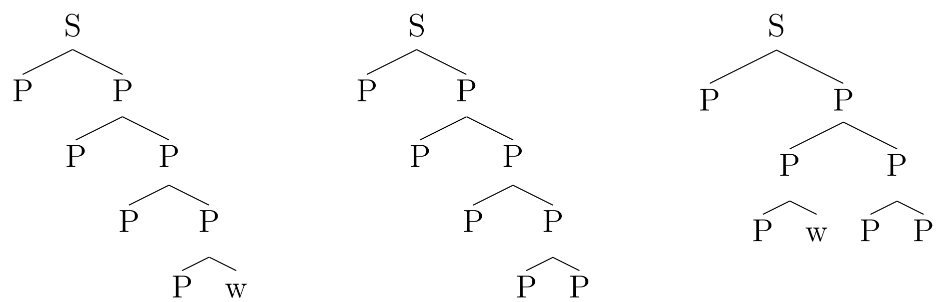


**Example 7.** Consider the transducer $M$ constructed as follows. We let $Q = \{q_w, q_p, q_S\}$, $\Sigma = \{w, P, S\}$, and $F = \{q_S\}$. The transition and ouput functions are given below.

- $\delta(\lambda, P) = q_p$
- $\delta(\lambda, w) = q_w$
- $\delta(q_p q_p, P) = q_p$
- $\delta(q_w q_p, P) = q_w$
- $\delta(q_p q_w, P) = q_w$
- $\delta(q_p q_w, S) = q_S$
- $\delta(q_w q_p, S) = q_S$
- $\delta(q_p q_p, S) = q_S$

- $\Omega(\lambda, P) = P[\ ]$
- $\Omega(\lambda, w) = w[\ ]$
- $\Omega(q_p q_p, P) = P[x_1 x_2]$
- $\Omega(q_w q_p, P) = P[x_1 x_2]$
- $\Omega(q_p q_w, P) = P[x_1 x_2]$
- $\Omega(q_p q_w, S) = S[\ w[\ ]\ S[x_1 x_2]\ ]$
- $\Omega(q_w q_p, S) = S[\ w[\ ]\ S[x_1 x_2]\ ]$
- $\Omega(q_p q_p, S) = S[x_1 x_2]$

Let us work out how $M$ transforms the trees below.




## 2.2 Deterministic Top-down Finite-state Tree Transducers

### 2.2.1 Orientation

This section is about deterministic bottom-up finite-state tree transducers. The term *finite-state* means that the amount of memory needed in the course of computation is independent of the size of the input. The term *deterministic* means there is single course of action the machine follows to compute its output. The term *transducer* means this machine solves *transformation problem*: given an input object $x$, what object $y$ is $x$ transformed into? The term *tree* means we are considering the transformation problem from trees to trees. The term *top-down* means that for each node $a$ in a tree, the computation transforms the node before transforming its children. This contrasts with *bottom-up* transducers which transform the children before transforming their parent. Visually, these terms make sense provided the root of the tree is at the top and branches of the tree move downward.

A definitive reference for finite-state automata for trees is freely available online. It is "Tree Automata Techniques and Applications" (TATA) (Comon *et al.*, 2007). The presentation here differs from the one there, as mentioned below.

### 2.2.2 Definitions

As before, we use variably leafed trees $\Sigma^T[X]$.

**Definition 13** (DTFTT)**.** *A* Deterministic Top-down Finite-state Acceptor (DTFTT) *is a tuple* $(Q, \Sigma_r, q_0, \delta)$ *where*

- $Q$ *is a finite set of states;*
- $\Sigma$ *is a finite alphabet;*
- $q_0 \in Q$ *is the initial state; and*
- $\delta : Q \times \Sigma \times \mathbb{N} \to Q^*$ *is* the transition function.
- $\Omega$ *is a function with domain* $Q \times \Sigma \times \mathbb{N}$ *and co-domain* $\Sigma^T[X]$.

*Generally, the pre-images of* $\delta$ *and* $\Omega$ *should coincide.*

We also define a new function "process" $\pi : Q \times \Sigma^T \to \Sigma^T$ which will process the tree and produce its output. It is defined as follows.

$$\begin{aligned} \pi(q, a[\,]) &= \Omega(q, a, 0) \\ \pi(q, a[t_1 \cdots t_n]) &= \Omega(q, a, n)\big\langle \pi(q_1, t_1) \cdots \pi(q_n, t_n)\big\rangle \\ &\quad \text{where } q_1 \cdots q_n = \delta(q, a, n) \end{aligned} \tag{2.2}$$

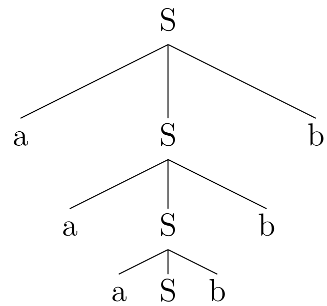
Intuitively, $\Omega$ transforms the root node into a variably leafed tree. The variables are replaced with the children of the root node. These children are also trees with states assigned by $\delta$. Then $\pi$ transforms each tree-child as well.

**Definition 14** (Tree-to-tree function of a DTFTT)**.** *The function defined by the transducer* $M$ *is* $\big\{(t, s) \mid t, s \in \Sigma^T, \pi(q_0, t) = s\big\}$*. If* $(t, s)$ *belongs to this set, we say* $M$ transduces $t$ *to* $s$ *and write* $M(t) = s$*.*

**Example 8.** Consider the transducer $M$ constructed as follows.

- $Q = \{q, q_S\}$
- $\Sigma = \{a, b, S\}$
- $q_0 = q_S$
- $\delta(q, a, 0) = \lambda$
- $\delta(q, b, 0) = \lambda$
- $\delta(q_S, S, 3) = q q_S q$
- $\delta(q_S, S, 2) = qq$
- $\Omega(q, a, 0) = a[\,]$
- $\Omega(q, b, 0) = b[\,]$
- $\Omega(q, S, 3) = S[x_3 x_2 x_1]$
- $\Omega(q, S, 2) = S[x_2 x_1]$

Let see how $M$ transforms the tree below.




**Exercise 2.** Recall the "wh-movement" example from before. Explain why this transformation *cannot* be computed by a deterministic top-down tree transducer.

## 2.3 Theorems about Deterministic Tree Transducers

**Theorem 6** (composition closure)**.** *The class of deterministic bottom-up transductions is closed under composition, but the class of top-down deterministic transductions is not.*

**Theorem 7** (Incomparable)**.** *The class of deterministic bottom-up transductions is incomparable with the the class of top-down deterministic transductions.*

This theorem is based on the same kind of examples which separated the left and rigth sequential functions. Let relations $U = (f^n a, f^n a) \mid n \in \mathbb{N} \cup (f^n b, g^n b)|n \in \mathbb{N}$ and $D = (ff^n a, ff^n a) \mid n \in \mathbb{N} \cup (gf^n a, gf^n b) \mid n \in \mathbb{N}$. $U$ is recognized by a DBFTT but not any DTFTT and $D$ is recognized by a DTFTT but not any DBFTT.

# Bibliography


Comon, H., M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. 2007. Tree automata techniques and applications. Available on: `http://tata.gforge.inria.fr/`. Release October, 12th 2007.

Davis, Martin D., and Elaine J. Weyuker. 1983. *Computability, Complexity and Languages*. Academic Press.

Harrison, Michael A. 1978. *Introduction to Formal Language Theory*. Addison-Wesley Publishing Company.

Hopcroft, John, Rajeev Motwani, and Jeffrey Ullman. 2001. *Introduction to Automata Theory, Languages, and Computation*. Boston, MA: Addison-Wesley.

Sipser, Michael. 1997. *Introduction to the Theory of Computation*. PWS Publishing Company.