1 Deterministic Top-down Finite-state Tree Acceptors

1.1 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 a single course of action the machine follows to compute its output. The term *acceptor* means this machine solves the 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 Definition

**Definition 1 (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 an initial state;
- \( \Sigma \) is a finite alphabet;
- \( \delta : Q \times \Sigma \times \mathbb{N} \rightarrow Q^* \) 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 \( \delta^* : Q \times \Sigma^T \rightarrow Q^* \) as follows.

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

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 2 (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 $L(A) = \{ t \in \Sigma^T | \delta^*(t) = \lambda \}$.

The use of the ‘L’ denotes “Language” as treesets are traditionally referred to as formal tree languages.

Example 1. Recall the example from last week which generates trees like

\[
\begin{aligned}
    &S \\
    &\text{a} \ 	ext{b} \\
\end{aligned}
\quad
\begin{aligned}
    &S \\
    &\text{a} \\
    &\text{S} \ 	ext{b} \\
    &\text{a} \ 	ext{b} \\
    &\text{S} \\
    &\text{a} \ 	ext{S} \ 	ext{b} \\
    &\text{a} \ 	ext{S} \ 	ext{b} \\
    &\text{S} \ 	ext{a} \\
    &\text{S} \ 	ext{b} \\
    &\text{S} \ 	ext{b} \\
    &\text{a} \ 	ext{b} \ 	ext{...}
\end{aligned}
\]

- $Q = \{q_a, q_b, q_S\}$
- $\Sigma = \{a, b, S\}$
- $q_0 = q_S$

$\delta(q_a, a, 0) = \lambda$
$\delta(q_b, b, 0) = \lambda$
$\delta(q_S, S, 3) = q_aqSq_b$
$\delta(q_S, S, 2) = q_aq_b$

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

Theorem 1. 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.

\[
\begin{aligned}
    &S \\
    &\text{a} \ 	ext{b} \\
\end{aligned}
\quad
\begin{aligned}
    &S \\
    &\text{b} \ 	ext{a} \\
\end{aligned}
\]

This is a recognizable treeset because the DBFTA below recognizes exactly these two trees and no others.
\[ Q = \{ q, q_S \} \]
\[ \Sigma = \{ a, b, S \} \]
\[ q_0 = q_S \]
\[ \delta(\lambda, a) = q_a \]
\[ \delta(\lambda, b) = q_b \]
\[ \delta(q_aq_b, S) = q_S \]
\[ \delta(q_bq_a, S) = q_S \]

Notice that this DBFTA fails on these two trees.

\[
\begin{array}{c}
S \\
  \overbrace{\overset{a}{\ldots}}^{a} \\
S \\
  \overbrace{\overset{b}{\ldots}}^{b}
\end{array}
\]

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 \( T \).

### 1.3 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 \( \delta(q, a, n) = q_1 \cdots q_n \).

### References