# Chapter 1

# Introduction: Strings and Trees

## 1.1 Strings and Trees

In this section, we define strings and trees of finite size inductively.

### 1.1.1 Strings

Informally, strings are sequences of symbols.

What are symbols? It is standard to assume a set of symbols called the *alphabet*. The Greek symbol $\Sigma$ is often used to represent the alphabet but people also use $S$, $A$, or anything else. The symbols can be anything: IPA letters, morphemes, words, part-of-speech categories. $\Sigma$ can be infinite in size, but we will usually consider it to be finite.

Formally, strings are defined inductively with an operation called *concatenation*. Concatenation is an operation, like addition, which build new strings from existing ones. For now, we will write it as "$\cdot$". *Inductive definitions* are defined in two parts: the *base case* and an *inductive case*.

**Definition 1** (Strings, version 1)**.**

**Base Case:** *If $a \in \Sigma$ then $a$ is a string.*
**Inductive Case:** *If $a \in \Sigma$ and $w$ is a string then $a \cdot (w)$ is a string.*

**Example 1.** Let $\Sigma = \{a, b, c\}$. Then the following are strings.

1. $a \cdot (b \cdot (c))$
2. $a \cdot (a \cdot (a))$
3. $a \cdot (b \cdot (c \cdot (c)))$

Frankly, writing all the parentheses and "$\cdot$" is cumbersome. So the above examples are much more readable if written as follows.

1. $abc$

2. *aaa*
3. *abcc*

Writing all the parentheses and "$\cdot$" is also unnecessary because the above definition provides a unique "derivation" for each string.

**Example 2.** Let $\Sigma = \{a, b, c\}$. We claim $w = a \cdot (b \cdot (c \cdot (c)))$ is a string. There is basically one way to show this. First we observe that $a \in \Sigma$ so whether $w$ is a string depends on whether $x = b \cdot (c \cdot (c))$ is a string by the inductive case. Next we observe that since $b \in \Sigma$ whether $x$ is a string depends on whether $y = c \cdot (c)$ is a string, again by the inductive case. Once more, since $c \in \Sigma$ whether $y$ is a string depends on whether $c$ is a string. Finally, by the base case $c$ is a string and so the dominoes fall: $y$ is a string so $x$ is a string and so $w$ is a string.

This unique derivability is useful in many ways. For instance, suppose we want to determine the length of a string. Here is how we can do it.

**Definition 2** (string length)**.** *The* length *of a string $w$, written $|w|$, is defined as follows. If there is $a \in \Sigma$ such that $w = a$ then $|w| = 1$. If not, then $w = a \cdot (x)$ where $x$ is some string and $a \in \Sigma$. In this case, $|w| = |x| + 1$.*

Note length is an inductive definition!

**Example 3.** What is the length of string $w = abcc$? Well, as before we see that $w = a \cdot x$ where $x = bcc$. Thus, $|w| = 1 + |bcc|$. What is the length of $bcc$? Well, $x = b \cdot y$ where $y = cc$. So now we have $|w| = 1 + (1 + |cc|)$. Since $y = c \cdot z$ where $z = c$ we have $|w| = 1 + (1 + (1 + |c|))$. Finally, by the *base case* we have $|w| = 1 + (1 + (1 + (1))) = 4$.

There is another way to define strings, which makes use of the so-called empty string. The empty string is usually written with one of the Greek letters $\epsilon$ or $\lambda$. It's just a matter of personal preference. The empty string is useful from a mathematical perspective in the same way the number zero is useful. Zero is a special number because for all numbers $x$ it is the case that $0 + x = x + 0 = x$. The empty string serves the same special purpose. It is the unique string with the following special property with respect to concatenation.[1]

$$\text{For all strings } w,\ \lambda \cdot w = w \cdot \lambda = w \tag{1.1}$$

With this concept, we can redefine strings as the following.

**Definition 3** (Strings, version 2)**.**

**Base Case:** *$\lambda$ is a string.*

[1] Technically, concatenation is not yet defined between strings since it is only defined for a string and a symbol. Our goal would be to ensure the property mentioned holds once concatenation is defined between strings.

**Inductive Case:** *If $a \in \Sigma$ and $w$ is a string then $a \cdot (w)$ is a string.*

As a technical matter, when we write the string *abc*, we literally mean the following structure: $a \cdot (b \cdot (c \cdot (\lambda)))$.

The definition for length can be redefined similarly.

We can now define concatenation between strings. First we define ReverseAppend, which takes two strings as arguments and returns a third string.

**Definition 4** (reverse append)**.** Reverse append *is a binary operation over strings, which we denote $\otimes_{\texttt{revapp}}$. You can also think of it as a function which takes two strings $w_1$ and $w_2$ as arguments and returns another string. Here is the base case. If $w_1 = \lambda$ then it returns $w_2$. So we can write $\lambda \otimes_{\texttt{revapp}} w = w$. Otherwise, there is $a \in \Sigma$ such that $w_1 = a \cdot (x)$ for some string $x$. In this case,* reverse append *returns $x \otimes_{\texttt{revapp}} a \cdot (w_2)$.*

**Exercise 1.** Work out what $abc \otimes_{\texttt{revapp}} def$ equals.

**Exercise 2.** What is $abc \otimes_{\texttt{revapp}} \lambda$? Write a definition for string reversal.

**Exercise 3.** Define the concatenation of two strings $w_1$ and $w_2$ using reverse append and string reversal. Prove this definition satisfies Equation 1.1.

The set of all strings of finite length, including the empty string, is written $\Sigma^*$. A *stringset* is a subset of $\Sigma^*$.

Stringsets are often called *formal languages*. Formal language theory is the study of stringsets. From a linguistic perspective, is is the study of string well-formedness.

### 1.1.2 Trees

Trees are like strings in that they are recursive structures. Informally, trees are structures with a single 'root' node which dominates a sequence of trees.

Formally, trees extend the dimensionality of string structures from 1 to 2. In addition to a relation for linear order, there is also a dominance relation.

Unlike strings, we will not posit "empty" trees because every tree has a root.

Like strings, we assume a set of symbols $\Sigma$. This is sometimes partioned into symbols of different types depending on whether the symbols can only occur at the leaves of the trees or whether they can dominate other trees. We don't make such a distinction here. In the following definition, parantheses are omitted for readability.

**Definition 5** (Trees)**.**

**Base Case (lin):** *$\lambda$ is the empty string of trees.*

**Base Case (dom):** *If $a \in \Sigma$ then $a[\lambda]$ is a tree.*

**Inductive Case:** *If $a \in \Sigma$ and $t_1 \cdot (t_2 \cdot (\ldots (t_n) \ldots))$ is a string of trees of length $n > 0$ then $a[t_1 \cdot (t_2 \cdot (\ldots (t_n) \ldots))]$ is a tree.*

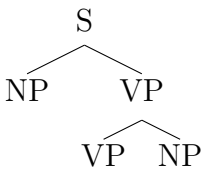
A tree $a[\lambda]$ is called a *leaf*. Note we typically write $a[\ ]$ and $a[t_1 t_2 \ldots t_n]$ for readability. Here are some examples.

**Example 4.** Let $\Sigma = \{\text{NP, VP, S}\}$. Then the following are trees.

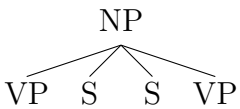
1. S[ NP[ ] VP[ VP[ ] NP[ ] ] ]
2. NP[ VP[ ] S[ ] S[ ] VP[ ] ]
3. NP[ NP[ NP[ ] VP[ ] S[ ] ] ]

We might draw these structures as follows.

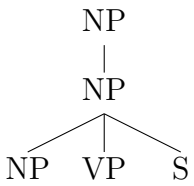
(1)




(2)




(3)




Regarding the tree in (1), its leaves are NP, VP, and NP.

Note that the expression "a string of trees" in the definition of trees implies that our alphabet for strings is all the trees. Since we are defining trees, this may seem a bit circular. The key to resolving this circularity is to interleave the definition of the alphabet of the strings with the definition of trees in a zig-zag fashion. First we apply the inductive case for trees *once*, then we use those trees as an alphabet to define some strings of trees. Then we go back to trees and apply the inductive case again, which yields more trees which we can use to enlarge our set of strings and so on. We are not going to go through the technical details here.

As before, we can now write definitions to get information about trees. For instance here is a definition which gives us the number of non-empty nodes in the tree.

**Definition 6.** *The* size *of a tree $t$, written $|t|$, is defined as follows. If there is some $a \in \Sigma$ such that $t = a[\ ]$ then its size is 1. If not, then $t = a[t_1 t_2 \ldots t_n]$ where $a \in \Sigma$ and each $t_i$ is a tree. Then $|t| = 1 + |t_1| + |t_2| + \ldots + |t_n|$.*

**Exercise 4.** Using the above definition, calculate the size of the trees (1)-(3) above. Write out the calculation explicitly.

Here is a definition for the *depth* of a tree.

**Definition 7.** *The* depth *of a tree $t$, written $\texttt{depth}(t)$, is defined as follows. If there is some $a \in \Sigma$ such that $t = a[\ ]$ then its depth is 0. If not, then $t = a[t_1 t_2 \ldots t_n]$ where $a \in \Sigma$ and each $t_i$ is a tree. Then $\texttt{depth}(t) = 1 + \texttt{max}\big\{\texttt{depth}(t_1), \texttt{depth}(t_2), \ldots, \texttt{depth}(t_n)\big\}$ where* max *takes the largest number in the set.*

**Exercise 5.** The *yield* of a tree $t$, written $\texttt{yield}(t)$, maps a tree to a string of its leaves. For example let $t$ be the tree in (1) in Example 4 above. Then its yield is the string "NP VP NP".