Computerlinguistik

WS 2018/2019

Greg Kobele

October 19, 2018

1 Strings and Trees

In order to understand the mathematical and computational properties of linguistic representations and operations over them, we need first to have precise definitions of the objects in question. We will focus on *strings* and *trees*, as these are the most common kinds of representations found in linguistics.

1.1 Strings

A string is a formal object inspired by a written word. Informally, it is a sequence of objects. The precise nature of the objects is not really relevant; we can have strings of roman letters, of cyrillic letters, of numbers, of cars (waiting at a gas station), of people (waiting in line at the store), etc. This is important - we can't have strings without objects, but the nature of the objects is irrelevant for the properties of strings we are concerned with!

We will, building on the motivating inspiration of a written word, call the set of objects appearing in a string its *alphabet*, and will often use the Greek capital letters Σ (or Δ, or Γ) to name an alphabet. In principle, alphabets may be infinite, but we will be mostly interested in situations where they are finite. In contrast, we will only be interested in finite strings.

There are many ways of formalizing almost *everything*; strings are no different. One of the most fundamental things we would like to do with strings is to *concatenate* them. Intuitively, concatenating strings is like writing them down next to one another. For example, concatenating the string "obst" with the string "garten" gives the string "obstgarten". One important property of string concatenation is that it is insensitive to constituency; concatenating strings "abc" and "def" and "ghi" (in that order) gives rise to the string "abcdefghi", regardless of whether we first concatenate "abc" and

"def" (to form "abcdef") and then concatenate that with "ghi", or whether we concatenate "abc" to the result of concatenating "def" and "ghi" (forming "defghi"). Indicating concatenation of strings via a dot ($\cdot$), we have that $"abc" \cdot ("def" \cdot "ghi") = ("abc" \cdot "def") \cdot "ghi"$. (Binary) operations which have this property are called *associative*. An adequate formalization of strings will allow us to define concatenation.

1.1.1 Strings as arrays

A first definition of strings views them as an ordered set of *positions*, which contain symbols. We begin by identifying positions with numbers, starting at 0.[1] If a string has (say) 10 positions, these will be named 0,1,2,3,...,8,9. In other words, the set of positions in a string of length n will be $\{0, 1, \ldots, n-1\}$. It will be very convenient to have a simple notation for the set of all (natural) numbers less than another. We define:

Definition 1. *For any natural number $n \in$, let $[n] := \{k : k < n\}$.*

A string of length n over an alphabet Σ is an assignment of symbols in Σ to each position in the string. What this means is that each position should be associated with exactly one symbol; this notion of an assigment is made precise as a *function*.

Definition 2. *Given an alphabet Σ, and a natural number n, a string of length n over Σ is a function from $[n]$ to Σ.*

Viewing a string as an association between positions and symbols (a function from positions to symbols) means that we can speak of the symbol at a particular position in terms of the result of applying the function to that position. We can visualize an example as in figure 1.

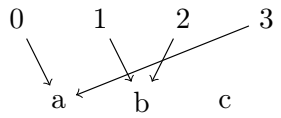

Figure 1: The string "abba" over the alphabet $\Sigma = \{a, b, c\}$

As a special case of our definition of a string, consider what happens when we have a string of length *0*. Then a string of length 0 is a function associating each position in [0] with a symbol. But, there *is* no position in [0];

[1] This has the unfortunate consequence of implying that the *first* position in a string is 0, the *second* is 1, etc.

$[0] = \emptyset$ is the empty set! This may seem puzzling at first: what should such a function do? It is easiest to see what to do in the context of a real-world example. Imagine that our job would be to give each subscriber the newspaper they ordered (i.e., we should implement a function from subscribers to newspapers). If we don't have any newspapers, then we are in trouble; we may be fired for not doing our job! But if we don't have any subscribers, then our job becomes trivial; we may as well relax on the beach, as we have finished before even starting! Returning to the function, we see that there is exactly one way of associating outputs with no inputs: do nothing. This is called the empty function, and in the world of strings, it is called the empty string, and is written with the Greek lower case ϵ.[2]

This discussion presupposes a notion of identity between strings, which we have not yet made explicit. Two strings are identical just in case the following holds:

1. they have the same set of positions
2. they have the same symbols in each of their positions

This can be written in a more mathematical idiom as follows.

Definition 3. *Given two strings u, v of length n over alphabet Σ, we say that $u = v$ iff for every $i \in [n]$ it holds that $u(i) = v(i)$.*

Consider how we might define string concatenation over these representations. As a concrete example, we may look at figure 2. Here we have representations of strings "ab" and "ba", which are concatenated to form the representation of the string "abba" which we saw above in figure 1.

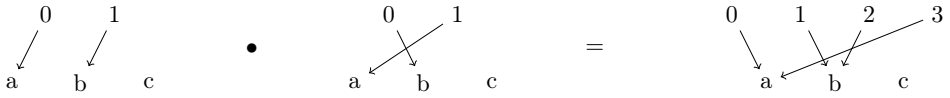

Figure 2: Concatenating "ab" with "ba" (over the alphabet $\Sigma = \{a, b, c\}$)

We can define a concatenation operation over these representations in the following way.

Definition 4. *Let Σ be an alphabet, and let u and v be strings over Σ of lengths m and n respectively. Then $u \cdot v$ is the string over Σ of length $m + n$, such that*

[2] In some places it is written with the Greek lower case λ.

1. *for each* $i \in [m]$, $(u \cdot v)(i) = u(i)$
2. *for each* $i \in [m+n] - [m]$, $(u \cdot v)(i) = v(i-m)$