Computerlinguistik

WS 2018/2019

Greg Kobele

October 23, 2018

1 Strings as lists

The position-based formalization of strings given above is correct, and everything we are interested in doing can be carried out in these terms. However, just because something is *true* doesn't make it *useful*! As a concrete example, we have a conception of natural numbers, which is independent of how we represent them.[1] Using a decimal representation of numbers allows us to perform certain operations, like multiplication by 10, very easily: just add a 0 to the end of the representation. Other operations, like multiplication by 2, require actual work to compute. In contrast, a binary representation of the same makes multiplication by 2 a matter of adding a 0 to the end of the representation, but has the trade-off of making multiplication by 10 more difficult. Both representations of natural numbers (and infinitely many more!) are correct, but, depending on what we would like to do with them, one may be more useful than the other.

Accordingly, we here present a different formalization of (and representation for) strings, which views them as being constructed from simpler pieces. We will say that a non-empty string contains a first symbol, and the rest of the string. In this way, larger strings can be built up out of smaller strings. Because we need to make reference to other strings when defining strings, we define at once the *set* of all strings over a certain alphabet.

Definition 1. Given an alphabet Σ, we define the strings over Σ as follows.

1. `[]` is a string over Σ
2. if w is a string over Σ, and $a \in \Sigma$, then $(a : w)$ is a string over Σ

[1] This might be given by the Peano axioms, for example.

implicit in the above definition is the further clause:

3. nothing else is a string over Σ

This definition tells us that a string (over Σ) can have exactly one of two general shapes; it can be empty, in which case it is $[]$, or non-empty, in which case it is of the form $a:w$ for some symbol 'a' and string w.[2] It gives us therefore not only a definition of strings, but also a procedure for determining whether something is a string over Σ.

Example 1. We attempt to verify that "abba" is indeed a string over $\{a, b, c\}$. Note that it is of the form $a:w$ (where $w =$ "bba"), and so is a string over $\{a, b, c\}$ iff $a \in \{a, b, c\}$ (which it is) and w is a string over $\{a, b, c\}$. So whether "abba" is a string depends on whether "bba" is a string. And "bba" (recall, it is of the form $b:w$, where $w =$ "ba") is a string iff $b \in \{a, b, c\}$ and "ba" is a string. Similarly, "ba" is a string iff "a" is, and "a" is a string iff $[]$ is. But $[]$ **is** a string (it says so in the definition). And so then "a" is, and then "ba" is, and then "bba" is, and then "abba" is, as we wanted to determine.

Thus, if we want to define a function over all strings, we need only to say what it does in these two circumstances. We call this a recursive definition. As an example, we define a function associating with each string its length, which should tell us how many symbols occur in it. Clearly, the length of the empty string, $[]$, is 0. An if we want to know the length of a string of the form $a:w$, it is one greater than the length of w.

Definition 2. The length of a string w, written $|w|$, is given inductively as follows.

- $|[]| = 0$
- $|a:w| = 1 + |w|$

What is unique about recursive definitions is that they define something in terms of itself.[3] That is, the definition appears *circular*. The naïve worry about circular definitions (such as of a word in a dictionary) is that you will be shunted from one place to another and back, and never make any progress. Not all definitions which refer to themselves are circular in this

[2] It is inconvenient to write $a:b:b:a:[]$ for the string "abba". We will simply write "abba" and convert in our minds between the simpler, orthographical writing convention and the official formal representation.

[3] We did this already in the definition of the set of strings over Σ.

sense, however! One important aspect of the definition of lengh given above is that the size of the argument given to the function we are defining is steadlily decreasing (w is shorter than $a : w$); as all strings are finite, this means that our length function is guaranteed to eventually reach its base case, where $w = \texttt{[]}$, and give us a concrete (i.e. non-self-referential) answer. We can illustrate the workings of this definition via an example.

Example 2. We trace through the action of the length function on the string *abba*.

$$\begin{aligned} |a : b : b : a : \texttt{[]}| &= 1 + |b : b : a : \texttt{[]}| \\ &= 1 + 1 + |b : a : \texttt{[]}| \\ &= 1 + 1 + 1 + |a : \texttt{[]}| \\ &= 1 + 1 + 1 + 1 + |\texttt{[]}| \\ &= 1 + 1 + 1 + 1 + 0 \\ &= 4 \end{aligned}$$