A Crash-Course on Notation

There are innumerable books available about automata and formal languages. Unfortunately, they often use their own notational conventions and their own symbols. This handout is meant to clarify some of the notation I will use.

Characters and Strings

The most important concept in this module are strings. Strings are composed of ***characters***. While characters are surely a familiar concept, we will make one subtle distinction in this module. If we want to refer to concrete characters, like `a`, `b` and so on, we use a typewriter font. Accordingly if we want to refer to the concrete characters of my email address we shall write

```
christian.urban@kcl.ac.uk
```

If we also need to explicitly indicate the "space" character, we write $\llcorner$. For example

```
hello␣world
```

But often we do not care which particular characters we use. In such cases we use the italic font and write a, b and so on for characters. Therefore if we need a representative string, we might write

$$abracadabra \tag{1}$$

In this string, we do not really care what the characters stand for, except we do care about the fact that for example the character a is not equal to b and so on.

An ***alphabet*** is a (non-empty) finite set of characters. Often the letter Σ is used to refer to an alphabet. For example the ASCII characters `a` to `z` form an alphabet. The digits 0 to 9 are another alphabet. The Greek letters α to ω also form an alphabet. If nothing else is specified, we usually assume the alphabet consists of just the lower-case letters a, b, …, z. Sometimes, however, we explicitly want to restrict strings to contain only the letters a and b, for example. In this case we will state that the alphabet is the set $\{a, b\}$.

Strings are lists of characters. Unfortunately, there are many ways how we can write down strings. In programming languages, they are usually written as "*hello*" where the double quotes indicate that we are dealing with a string. But since we regard strings as lists of characters we could also write this string as

$$[h, e, l, l, o]$$

The important point is that we can always decompose such strings. For example, we will often consider the first character of a string, say h, and the "rest" of a string say "*ello*" when making definitions about strings. There are also some

subtleties with the empty string, sometimes written as "" but also as the empty list of characters $[]$.[1]

Two strings, say s_1 and s_2, can be ***concatenated***, which we write as $s_1 @ s_2$. Suppose we are given two strings "*foo*" and "*bar*", then their concatenation, writen "*foo*" @ "*bar*", gives "*foobar*". Often we will simplify our life and just drop the double quotes whenever it is clear we are talking about strings, writing as already in (1) just *foo*, *bar*, *foobar* or *foo* @ *bar*.

Some simple properties of string concatenation hold. For example the concatenation operation is *associative*, meaning

$$(s_1 @ s_2) @ s_3 = s_1 @ (s_2 @ s_3)$$

are always equal strings. The empty string behaves like a unit element, therefore

$$s @ [] = [] @ s = s$$

Occasionally we will use the notation a^n for strings, which stands for the string of n repeated as. So $a^n b^n$ is a string that has as many as as bs.

While for us strings are just lists of characters, programming languages often differentiate between the two concepts. In Scala, for example, there is the type of `String` and the type of lists of characters, `List[Char]`. They are not the same and we need to explicitly coerce elements between the two types, for example

```
scala> "abc".toList
res01: List[Char] = List(a, b, c)
```

Sets and Languages

We will use the familiar operations $\cup$, $\cap$, $\subset$ and $\subseteq$ for sets. For the empty set we will either write $\varnothing$ or $\{\}$. The set containing the natural numbers 1, 2 and 3, for example, we will write with curly braces as

$$\{1, 2, 3\}$$

The notation $\in$ means *element of*, so $1 \in \{1, 2, 3\}$ is true and $3 \in \{1, 2, 3\}$ is false. Sets can potentially have infinitely many elements. For example the set of all natural numbers $\{0, 1, 2, \ldots\}$ is infinite. This set is often also abbreviated as $\mathbb{N}$. We can define sets by giving all elements, for example $\{0, 1\}$, but also by ***set comprehensions***. For example the set of all even natural numbers can be defined as

$$\{n \mid n \in \mathbb{N} \wedge n \text{ is even}\}$$

Though silly, but the set $\{0, 1, 2\}$ could also be defined by the following set comprehension

[1] In the literature you can also often find that ε or λ is used to represent the empty string.

$$\{n \mid n^2 < 9 \wedge n \in \mathbb{N}\}$$

Notice that set comprehensions could be used to define set union, intersection and difference:

$$\begin{array}{rcl} A \cup B & \stackrel{\text{def}}{=} & \{x \mid x \in A \vee x \in B\} \\ A \cap B & \stackrel{\text{def}}{=} & \{x \mid x \in A \wedge x \in B\} \\ A \backslash B & \stackrel{\text{def}}{=} & \{x \mid x \in A \wedge x \notin B\} \end{array}$$

In general set comprehensions are of the form $\{a \mid P\}$ which stands for the set of all elements a (from some set) for which some property P holds.

For defining sets, we will also often use the notion of the "big union". An example is as follows:

$$\bigcup_{0 \leq n} \{n^2, n^2 + 1\} \tag{2}$$

which is the set of all squares and their immediate successors, so

$$\{0, 1, 2, 4, 5, 9, 10, 16, 17, \ldots\}$$

A big union is a sequence of unions which are indexed typically by a natural number. So the big union in (2) could equally be written as

$$\{0, 1\} \cup \{1, 2\} \cup \{4, 5\} \cup \{9, 10\} \cup \ldots$$

but using the big union notation is more concise.

An important notion in this module are ***languages***, which are sets of strings. The main goal for us will be how to (formally) specify languages and to find out whether a string is in a language or not.[2] Note that the language containing the empty string $\{\texttt{""}\}$ is not equal to $\varnothing$, the empty language (or empty set): The former contains one element, namely `""` (also written $[\,]$), but the latter does not contain any element.

For languages we define the operation of ***language concatenation***, written like in the string case as $A@B$:

$$A@B \stackrel{\text{def}}{=} \{s_1@s_2 \mid s_1 \in A \wedge s_2 \in B\} \tag{3}$$

Be careful to understand the difference: the @ in $s_1@s_2$ is string concatenation, while $A@B$ refers to the concatenation of two languages (or sets of strings). As an example suppose $A = \{ab, ac\}$ and $B = \{zzz, qq, r\}$, then $A \,@\, B$ is the language

$$\{abzzz, abqq, abr, aczzz, acqq, acr\}$$

[2] You might wish to ponder whether this is in general a hard or easy problem, where hardness is meant in terms of Turing decidable, for example.

Recall the properties for string concatenation. For language concatenation we have the following properties

$$\begin{array}{ll} \text{associativity:} & (A@B)@C = A@(B@C) \\ \text{unit element:} & A @ \{[]\} = \{[]\} @ A = A \\ \text{zero element:} & A @ \varnothing = \varnothing @ A = \varnothing \end{array}$$

Note the difference in the last two lines: the empty set behaves like 0 for multiplication and the set $\{[]\}$ like 1 for multiplication.

Following the language concatenation, we can define a ***language power*** operation as follows:

$$\begin{array}{rcl} A^0 & \stackrel{\text{def}}{=} & \{[]\} \\ A^{n+1} & \stackrel{\text{def}}{=} & A @ A^n \end{array}$$

This definition is by recursion on natural numbers. Note carefully that the zero-case is not defined as the empty set, but the set containing the empty string. So no matter what the set A is, A^0 will always be $\{[]\}$. (There is another hint about a connection between the @-operation and multiplication: How is x^n defined and what is x^0?)

Next we can define the ***star operation*** for languages: A^* is the union of all powers of A, or short

$$A^* \stackrel{\text{def}}{=} \bigcup_{0 \leq n} A^n \tag{4}$$

This star operation is often also called *Kleene-star*. Unfolding the definition in (4) gives

$$A^0 \cup A^1 \cup A^2 \cup A^3 \cup \ldots$$

which is equal to

$$\{[]\} \cup A \cup A@A \cup A@A@A \cup \ldots$$

We can see that the empty string is always in A^*, no matter what A is. This is because $[] \in A^0$. To make sure you understand these definitions, I leave you to answer what $\{[]\}^*$ and $\varnothing^*$ are.

Recall that an alphabet is often referred to by the letter Σ. We can now write for the set of all strings over this alphabet Σ^*. In doing so we also include the empty string as a possible string over Σ. So if $\Sigma = \{a, b\}$, then Σ^* is

$$\{[], a, b, ab, ba, aaa, aab, aba, abb, baa, bab, \ldots\}$$

or in other words all strings containing as and bs only.