Handout 1

This course is about processing of strings. Lets start with what we mean by *string*. Strings are lists of characters drawn from an *alphabet*. If nothing else is specified, we usually assume the alphabet are letters a, b, ..., z and A, B, ... Z. Sometimes we explicitly restrict strings to only contain the letters a and b. Then we say the alphabet is the set $\{a, b\}$.

There are many ways how we write string. Since they are lists of characters we might write them as "*hello*" being enclosed by double quotes. This is a shorthand for the list

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

The important point is that we can always decompose strings. For example we often consider the first character of a string, say h, and the "rest" of a string "*ello*". There are also some subtleties with the empty string, sometimes written as "" or as the empty list of characters $[\,]$. Two strings, say s_1 and s_2, can be *concatenated* which is written as $s_1 @ s_2$. For example if we have the two strings "*foo*" and "*bar*", their concatenation gives "*foobar*".

We often need to talk about sets of strings. For example the set of all strings

$$\{\text{"", "}a\text{", "}b\text{", "}c\text{",..., "}z\text{", "}aa\text{", "}ab\text{", "}ac\text{", ..., "}aaa\text{", ...}\}$$

Any set of strings, not just the set of all strings, is often called a *language*. The idea behind this choice is that if we enumerate, say, all words/strings from a dictionary, like

$$\{\text{"}the\text{", "}of\text{", "}milk\text{", "}name\text{", "}antidisestablishmentarianism\text{", ...}\}$$

then we have essentially described the English language, or more precisely all strings that can be used in a sentence of the English language. French would be a different set of string, and so on. In the context of this course, a language might not necessarily make sense from a natural language perspective. For example the set of all strings from above is a language, as is the empty set (of strings). The empty set of strings is often written as $\varnothing$ or $\{\,\}$. Note that there is a difference between the empty set, or empty language, and the set, or language, that contains the empty string $\{""\}$: the former has no elements, the latter has one element.

As seen there are languages which contain infinitely many strings, like the set of all strings. The "natural" languages English, French and so on contain many but only finitely many strings (the ones listed in a good dictionary). It might be therefore surprising that the language consisting of all email addresses is infinite if we assume it is defined by the regular expression[1]

$$([a\text{-}z0\text{-}9\_.\text{-}]^+)@([a\text{-}z0\text{-}9.\text{-}]^+).([a\text{-}z.]^{\{2,6\}})$$

[1] See `http://goo.gl/5LoVX7`

The reason is that for example before the @-sign there can be any string you want if it is made up from letters, digits, underscore, dot and hyphen—there are infinitely many of those. Similarly the string after the @-sign can be any string. This does not mean that every string is an email address. For example

$$\textit{foo@bar.c}$$

is not, since the top-level-domains must be of length of at least two. Note that there is the convention that uppercase letters are treated in email-addresses as if they were lower-case.

Before we expand on the topic of regular expressions, let us review some operations on sets. We will use capital letters, such as A, B and so on, to stand for sets of strings. The union of two sets is written as usual as $A \cup B$. We also need to define the *concatenation* of two sets (of strings). This can be defined as

$$A@B \stackrel{\text{def}}{=} \{s_1@s_2 | s_1 \in A \wedge s_2 \in B\}$$

which essentially means take the first string from the set A and concatenate it with every string in the set B, then take the second string from A and so on. We also need to define the power of a set, written as A^n. This is defined inductively as follows

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

Finally we need the *star* of a set of strings, written A^*. This is defined as the union of every power of A^n for every $n \geq 0$. The mathematical notation for this operation is

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

This means the star of a set A contains always the empty string, one copy of every string in A, two copies and so on. In case $A = \{"a\}$ we have

$$A^* = \{"", "a", "aa", "aaa", \ldots\}$$

Be aware that these operations sometimes have non-intuitive properties, for example

$$\begin{array}{lll} A \cup \varnothing = A & A@B \neq B@A & \varnothing^* = \{""\} \\ A \cup A = A & A@\varnothing = \varnothing@A = \varnothing & \{""\}^* = \{""\} \\ A \cup B = B \cup A & A@\{""\} = \{""\}@A = A & A^1 = A \end{array}$$

Regular expressions are meant for conveniently describing languages...at least languages we are interested in in Computer Science. For example there is no

convenient regular expression for describing the English language short of enumerating all English words/strings like in a dictionary. But they seem useful for describing all permitted email addresses, as seen above.

Regular expressions are given by the following grammar:

$$
\begin{array}{lll}
r ::= \varnothing & & \text{null} \\
\mid \epsilon & & \text{empty string / "" / []} \\
\mid c & & \text{character} \\
\mid r_1 \cdot r_2 & & \text{sequence} \\
\mid r_1 + r_2 & & \text{alternative / choice} \\
\mid r^* & & \text{star (zero or more)}
\end{array}
$$

There are some subtleties you should be aware of. The letter c stands for any character from the alphabet. Second, we will use parentheses to disambiguate regular expressions. For example we will write $(r_1 + r_2)^*$, which is different from $r_1 + (r_2)^*$. The former means roughly zero or more times r_1 or r_2, while the latter means r_1 or zero or more times r_2. We should also write $(r_1 + r_2) + r_3$ which is a regular expression different from $r_1 + (r_2 + r_3)$, but in case of $+$ and $\cdot$ we actually do not care and just write $r_1 + r_2 + r_3$, or $r_1 \cdot r_2 \cdot r_3$, respectively. The reasons for this will become clear shortly. In the literature you will often find that the choice $r_1 + r_2$ is written as $r_1 \mid r_2$. In case of $\cdot$ we will even often omit it all together. For example the regular expression for email addresses is meant to be of the form

$$([\ldots])^+ \cdot @ \cdot ([\ldots])^+ \cdot \cdot \cdot \ldots$$

meaning first comes a name (specified by the regular expression $([\ldots])^+$), then an @-sign, then a domain name (specified by the regular expression $([\ldots])^+$), then a top-level domain. Similarly if we want to specify the regular expression for the string "*hello*" we should write

$$h \cdot e \cdot l \cdot l \cdot o$$

but often just write *hello*.

Another source of confusion might be that we use the term *regular expressions* for the ones used in "theory" and the ones in "practice". In this course by default we refer to the regular expressions defined by the grammar above. In "practice" we often use r^+ for one or more times, $\backslash d$ to stand for a digit, $r^?$ for an optional regular expression, or ranges such as $[a\text{ - }z]$ to stand for any lower case letter from a to z. They are mere convenience as they can be seen as shorthand for

$$
\begin{array}{rcl}
r^+ & \mapsto & r \cdot r^* \\
r^? & \mapsto & \epsilon + r \\
\backslash d & \mapsto & 0 + 1 + 2 + \ldots + 9 \\
[a\text{ - }z] & \mapsto & a + b + \ldots + z
\end{array}
$$

We will see later that the *not*-regular-expression can also be seen as convenience. This regular expression is supposed to stand for every string *not* match by a regular expression. We will write such regular expressions as $\underline{r}$. While being "convenience" it is often not so clear what the shorthand for these kind of regular expressions is.

So far we have only considered informally what the *meaning* of a regular expression is. Formally we associate with every regular expression a set of strings which are matched by this regular expression. This can be formally defined as

$$\begin{array}{rcl} L(\varnothing) & \stackrel{\text{def}}{=} & \{\,\} \\ L(\epsilon) & \stackrel{\text{def}}{=} & \{""\} \\ L(c) & \stackrel{\text{def}}{=} & \{"c"\} \\ L(r_1 + r_2) & \stackrel{\text{def}}{=} & L(r_1) \cup L(r_2) \\ L(r_1 \cdot r_2) & \stackrel{\text{def}}{=} & \{s_1 @ s_2 | s_1 \in L(r_1) \wedge s_2 \in L(r_2)\} \\ L(r^*) & \stackrel{\text{def}}{=} & \bigcup_{n \geq 0} L(r)^n \end{array}$$

This means we can now precisely state what the meaning, for example, of the regular expression $h \cdot e \cdot l \cdot l \cdot o$ is, namely $L(h \cdot e \cdot l \cdot l \cdot o) = \{"hello"\}$. Similarly if we have the choice $a + b$, the meaning is $L(a + b) = \{"a", "b"\}$, namely the only two strings which can possibly be matched by this choice.

The point of this definition is that we can now precisely specify when a string is matched by a regular expression, namely a string, say s, is matched by a regular expression, say r, if and only if $s \in L(r)$. In fact we will write a program *match* that takes any string s and any regular expression r as argument and returns *yes*, if $s \in L(r)$ and *no*, if $s \notin L(r)$. We leave this for the next lecture.