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 short-hand 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 $[\,]$.

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

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

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

$$\{"the", "of", "milk", "name", "antidisestablishmentarianism", \ldots\}$$

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]

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

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

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

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

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.

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:

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

There are some subtleties you should be aware of. First, 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

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

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.