Lecture 8: Regular Languages

During the previous lectures, we have encountered many examples of sets. Most of these sets are infinite. How can we talk about sets like, valid US telephone numbers? A phone number in the US is always written as (412)421-8558. This means a telephone number starts always a prefix in parenthesis the so called area code. A telephone number is then 7 digits. and there is a - symbol after the third digit.

How can we convince a machine to tell us some given telephone number is a US phone number or not. Consider for example 33.12.43.23, which clearly is not a US number. For starters, it doesn't even start with a parenthesis.

Let's tackle to problem form a different angle. Sets. How do we define the set of all US telephone numbers? First of all, we need to figure what a phone number really is: it is not a number, because it contains funny characters like parentheses and even a minus sign. A phone number is nothing else but a list of symbols.

The set of all symbols that may occur in a telephone number is easily enumerated, it forms a set. {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, (,), -} This set is also called the *alphabet*. The set of all lists over our alphabet contains a lot of lists (let's just call them strings.) We will write "(800)555-1212" for strings, please note the quotes. Many more then we are interested in. To restrict the set, we will use something that is also called a regular expression. A regular expression is nothing but a description for a set of certain elements. For example the regular expression "8" simply represents the set of all strings of the form "8", of which there is only one.

Let's make this representation relationship a bit more precise and start introducing regular expressions: "8" is an example of a regular expression. With every regular expression R we associate a set of strings, called *the language* of R written as $\mathcal{L}(R)$. For example, $\mathcal{L}(\text{"8"}) = \{\text{"8"}\}$ and more general $\mathcal{L}(c) = c$ for every symbol in the alphabet c.

A digit is a one character string, such as "0", "1", ... "9". Wouldn't it be nice if we could talk about the set of digits in terms of a regular expression? We would need to define a Digit, such that

$$\mathcal{L}(\text{Digit}) = \{\text{"0"}, \text{"1"}, \ldots, \text{"9"}\} = \mathcal{L}(\text{"0"}) \cup \mathcal{L}(\text{"1"}) \cup \ldots \cup \mathcal{L}(\text{"9"})$$

and there is: $\mathcal{L}(\text{Digit}) = \mathcal{L}(\text{"0"} + \text{"1"} + \ldots + \text{"9"})$. Where + is a connective defining regular expressions. This connective is also called alternative: Digit = "0" + "1" + ... + "9".

An area code is always three digits in parenthesis. So far, the language sets that we have defined only contain one character strings. We need to be able to talk about sets of appended strings. For example, we need to define a regular expression TwoDigits, such that

$$\mathcal{L}(\text{TwoDigits}) = \{x@y \mid x \in \mathcal{L}(\text{Digit}) \wedge y \in \mathcal{L}(\text{Digit})\}$$

We define the regular expression TwoDigits = Digit×Digit, just call the operator concatenation.

Now it is easy to write down an expression for US phone numbers:

$$\begin{array}{lcl} \text{AreaCode} & = & \text{Digit} \times \text{Digit} \times \text{Digit} \\ \text{Exchange} & = & \text{Digit} \times \text{Digit} \times \text{Digit} \\ \text{Subscriber} & = & \text{Digit} \times \text{Digit} \times \text{Digit} \times \text{Digit} \\ \text{PhoneNumber} & = & \text{"("} \times \text{AreaCode} \times \text{")"} \times \text{Exchange} \times \text{"}-\text{"} \times \text{Subscriber} \end{array}$$

$+$ and $\times$ almost look like arithemetic connectives, right? In mathematics we have seen that 0 is the neutral element for addition and 1 for multiplcation (meaning, that $0 + x = x$ and $1 \times x = x$). Does it make sense to consider 0 and 1 also as regular expressions, and if yes how? If 0 ought to be the neutral element for alternative, then how shall we define the language of 0 such that $\mathcal{L}(0 + R) = \mathcal{L}(R)$? The answer can be easily derived:

$$\mathcal{L}(0 + R) = \mathcal{L}(0) \cup \mathcal{L}(R) = \mathcal{L}(R)$$

and thus $\mathcal{L}(0) = \emptyset$. Something very similar holds for 1: If 1 ought to be the neutral element for concatentation, then how shall we define the language of 1, such that $\mathcal{L}(1 \times R) = \mathcal{L}(R)$? Also here, the answer can be easily derived:

$$\mathcal{L}(1 \times R) = \{x@y \mid x \in \mathcal{L}(1) \wedge y \in \mathcal{L}(R)\} = \mathcal{L}(R)$$

and thus $\mathcal{L}(1) = \{\epsilon\}$, the set that only contains the empty string.

The final connective, we must meet is that of repetition, also called the Kleene star. In the example above, we have seen that digit is used three and four times in sequence. Could we also express arbitrarily long sequences of strings of digits using regular expressions? Yes we can, but we need to define it using the star:

$$\text{ArbitraySequenceOfDigits} = \text{Digit}^*$$

where $R^* = 1 + R + R \times R + R \times R \times R \ldots$ The following definition will be useful for the subsequent definitions:

Definition 40 (Pointwise string concatenation) *Let $S \subset \mathbb{L}$ and $T \subset \mathbb{L}$ then the set of all pairwise string concatentations $S \cdot T$ is defined as follows: $s \in S \cdot T$ if and only if $\exists s_1 \in S . \exists s_2 \in T . s = s_1 s_2$.*

In summary, we define a regular expression by the following grammar: $R ::= c \mid 0 \mid 1 \mid R_1 \times R_2 \mid R_1 + R_2 \mid R^*$ for the trees describing regular expressions. c is a symbol from the alphabet. Let's us define the language of regular expressions.

$$\begin{array}{lcl} \mathcal{L}(c) & = & \{c\} \\ \mathcal{L}(0) & = & \emptyset \\ \mathcal{L}(1) & = & \{\epsilon\} \\ \mathcal{L}(R_1 \times R_2) & = & \mathcal{L}(R_1) \cdot \mathcal{L}(R_2) \\ \mathcal{L}(R_1 + R_2) & = & \mathcal{L}(R_1) \cup \mathcal{L}(R_2) \\ \mathcal{L}(R^*) & = & \mathcal{L}(1 + R \times R^*) \end{array}$$

With regular expression, we can express many sets very succinctly. In particular, sometimes we encounter two regular expressions whose languages coincide, we define:

Definition 41 (Equivalence of regular expression) *Let R_1, R_2 be regular expressions. We say that $R_1 = R_2$ if and only if $\mathcal{L}(R_1) = \mathcal{L}(R_2)$.*

Lemma 42 (Distributivity)

$$(R \cup S) \cdot T = (R \cdot T) \cup (S \cdot T)$$

Proof: Direct, but we must show two directions. First direction: let $s \in (R \cup S) \cdot T$. By definition above, there exists s_1, s_2, such that $s = s_1 @ s_2$ and $s\_1 \in R \cup S$ and $s_2 \in T$. Case 1: $s\_1 \in R$ then $s = s\_1 @ s\_2 \in R \cdot T \subset (R \cdot T) \cup (S \cdot T)$. Case 2: $s\_1 \in S$ then $s = s\_1 @ s\_2 \in S \cdot T \subset (R \cdot T) \cup (S \cdot T)$. Second direction: let $s \in (R \cdot T) \cup (S \cdot T)$. Case 1: $s \in (R \cdot T)$. By definition above, there exists a s_1, s_2, such that $s = s_1 \cdot s_2$, such that $s_1 \in R$ and $s_2 \in T$. Therefore $s_1 \in R \subset R \cup S$ and hence $s = s_1 @ s_2 \in (R \cup S) \cdot T$. Case 2: Analogous. □