3 Regular expressions

Given an alphabet \( \Sigma \) a language is a set of words \( L \subseteq \Sigma^* \). So far we were able to describe languages either by using set theory (i.e. enumeration or comprehension) or by an automaton. In this section we shall introduce regular expressions as an elegant and concise way to describe languages. We shall see that the languages definable by regular expressions are precisely the same as those accepted by deterministic or non-deterministic finite automata. These languages are called regular languages or (according to the Chomsky hierarchy) Type 3 languages.

As already mentioned in the introduction regular expressions are used to define patterns in programs such as grep. grep gets as an argument a regular expression and then filters out all those lines from a file which match the regular expression, where matching means that the line contains a substring which is in the language assigned to the regular expression. It is interesting to note that even in the case when we search for a specific word (this is a special case of a regular expression) programs like grep are more efficient than a naive implementation of word search.

To find out more about grep have a look at the UNIX manual page and play around with grep. Note that the syntax grep uses is slightly different from the one we use here. grep also use some convenient shorthands which are not relevant for a theoretical analysis of regular expressions because they do not extend the class of languages.

3.1 What are regular expressions?

We assume as given an alphabet \( \Sigma \) (e.g. \( \Sigma = \{a, b, c, \ldots, z\} \)) and define the syntax of regular expressions (over \( \Sigma \))

1. \( \emptyset \) is a regular expression.
2. \( \epsilon \) is a regular expression.
3. For each \( x \in \Sigma \), \( x \) is a regular expression. E.g. in the example all small letters are regular expression. We use boldface to emphasize the difference between the symbol \( a \) and the regular expression \( a \).
4. If \( E \) and \( F \) are regular expressions then \( E + F \) is a regular expression.
5. If \( E \) and \( F \) are regular expressions then \( \epsilon F \) (i.e. just one after the other) is a regular expression.
6. If \( E \) is a regular expression then \( E^* \) is a regular expression.
7. If \( E \) is a regular expression then \((E)\) is a regular expression.

These are all regular expressions.

Here are some examples for regular expressions:

- \( \epsilon \)
- \( \text{hallo} \)
- \( \text{hallo } + \text{hello} \)
- \( b(a + c) \text{Hello} \)
- \( a\epsilon \)
- \( (\epsilon + b)(ab)^*(\epsilon + a) \)

As in arithmetic they are some conventions how to read regular expressions:

- \( \epsilon \) binds stronger then sequence and \( + \). E.g. we read \( ab \) as \( \epsilon \).
- Sequencing binds stronger than \( + \). E.g. we read \( ab + cd \) as \( (ab) + (bc) \).

To enforce another reading we have to use parentheses as in \( a(b + c)d \).

To find out more about grep have a look at the UNIX manual page and play around with grep. Note that the syntax grep uses is slightly different from the one we use here. grep also use some convenient shorthands which are not relevant for a theoretical analysis of regular expressions because they do not extend the class of languages.

3.2 The meaning of regular expressions

We now know what regular expressions are but what do they mean? For this purpose, we shall first define an operation on languages called the Kleene star. Given a language \( L \subseteq \Sigma^* \) we define

\[
L^* = \{ w_1w_2 \ldots w_{n-1} \mid n \in \mathbb{N} \land \forall i \in n \land w_i \in L \}
\]

Intuitively, \( L^* \) contains all the words which can be formed by concatenating an arbitrary number of words in \( L \). This includes the empty word since the number may be 0.

As an example consider \( L = \{a, b\} \subseteq \{a, b\}^* \):

\[
L^* = \{ \epsilon, a, ab, aab, aaba, aabaa, \ldots \}
\]

You should notice that we use the same symbol as in \( \Sigma^* \) but there is a subtle difference: \( \Sigma \) is a set of symbols but \( L \subseteq \Sigma^* \) is a set of words.

Alternatively (and more abstractly) one may describe \( L^* \) as the least language (wrt \( \subseteq \)) which contains \( L \) and the empty word and is closed under concatenation:

\[
w \in L^* \land v \in L^* \implies uv \in L^*
\]

We now define the semantics of regular expressions. To each regular expression \( E \) over \( \Sigma \) we assign a language \( L(E) \subseteq \Sigma^* \). We do this by induction over the definition of the syntax:

1. \( L(\emptyset) = \emptyset \)
2. \( L(x) = \{ x \} \)
3. \( L(\epsilon) = \{ \epsilon \} \)
4. \( L(E + F) = L(E) \cup L(F) \)
5. \( L(\epsilon E) = \{ \epsilon \} \cup L(E) \)
6. \( L(E^*) = L(E)^* \)
7. \( L((E)) = L(E) \)
we know to make this we obtain: we conclude we get:}

Subtle points: in 1. the symbol \( \emptyset \) may be used as a regular expression (as in \( L(\emptyset) \)) or the empty set (\( \emptyset = \{ \} \)). Similarly, \( \epsilon \) in 2. may be a regular expression or a word, in 6. \( * \) may be used to construct regular expressions or it is an operation on languages. Which alternative we mean becomes only clear from the context, there is no generally agreed mathematical notation \(^1\) to make this difference explicit.

Let us now calculate what the examples of regular expressions from the previous section mean, i.e. what are the languages they define:

\[ L(\epsilon) = \{ \epsilon \} \]

By 2.

hallo

Let’s just look at \( L(ha) \). We know from 3:

\[ L(h) = \{ h \} \]

\[ L(a) = \{ a \} \]

Hence by 5:

\[ L(ha) = \{ uv | u \in \{ h \} \land v \in \{ a \} \} \]

= \{ ha \}

Continuing the same reasoning we obtain:

\[ L(hallo) = \{ hallo \} \]

hallo + hello

From the previous point we know that:

\[ L(hallo) = \{ hallo \} \]

\[ L(hello) = \{ hello \} \]

Hence by using 4 we get:

\[ L(hallo + hello) = \{ hallo \} \cup \{ hello \} \]

= \{ hallo, hello \}

h(a + e)llo

Using 3 and 4 we know

\[ L(a + e) = \{ a, e \} \]

Hence using 5 we obtain:

\[ L(h(a + e)llo) = \{ uvw | u \in L(h) \land v \in L(a + e) \land w \in L(llo) \} \]

= \{ uvw | u \in \{ h \} \land v \in \{ a, e \} \land w \in \{ llo \} \}

= \{ hallo, hello \}

\(^1\)This is different in programming, e.g. in JAVA we use "..." to signal that we mean things literally.

\( a^*b^* \)

Let us introduce the following notation:

\[ w^* = w^\infty \quad \forall w \in \Sigma \]

Now using 6 we know that

\[ L(a^*) = \{ w_0 w_1 \ldots w_{n-1} | n \in \mathbb{N} \land \forall i < n, w_i \in L(a) \} \]

= \{ w_0 w_1 \ldots w_{n-1} | n \in \mathbb{N} \land \forall i < n, w_i \in \{ a \} \}

= \{ a^* | n \in \mathbb{N} \}

and hence using 5 we conclude

\[ L(a^*b^*) = \{ w | u \in L(a^*) \land v \in L(b^*) \} \]

= \{ uv | u \in \{ a^n | n \in \mathbb{N} \} \land v \in \{ b^m | m \in \mathbb{N} \} \}

= \{ a^*b^* | n, m \in \mathbb{N} \}

i.e. \( L(a^*b^*) \) is the set of all words which start with a (possibly empty) sequence of \( a \) followed by a (possibly empty) sequence of \( b \).

\( (\epsilon + b)(ab)^*(\epsilon + a) \)

Let’s analyze the parts:

\[ L(\epsilon + b) = \{ \epsilon, b \} \]

\[ L((ab)^*) = \{ ab^* | i \in \mathbb{N} \} \]

\[ L(\epsilon + b) = \{ \epsilon, b \} \]

Hence, we have

\[ L((\epsilon + b)(ab)^*(\epsilon + a)) = \{ u(ab)^*v | u \in \{ \epsilon, b \} \land i \in \mathbb{N} \land a^*v \in \{ \epsilon, b \} \}

In english: \( L((\epsilon + b)(ab)^*(\epsilon + a)) \) is the set of (possibly empty) sequences of interchanging as and bs.

3.3 Translating regular expressions to NFAs

Theorem 3.1 For each regular expression \( E \) we can construct an NFA \( N(E) \)

s.t. \( L(N(E)) = L(E) \), i.e. the automaton accepts the language described by the regular expression.

Proof:

We do this again by induction on the syntax of regular expressions:

1. \( N(\emptyset) \):

\[ L(\emptyset) = \{ \} \]

16
which will reject everything (it has got no final states) and hence

\[ L(N(\emptyset)) = \emptyset = L(\emptyset) \]

2. \(N(\epsilon)\):

This automaton accepts the empty word but rejects everything else, hence:

\[ L(N(\epsilon)) = \{\epsilon\} = L(\epsilon) \]

3. \(N(x)\):

This automaton only accepts the word \(x\), hence:

\[ L(N(x)) = \{x\} = L(x) \]

4. \(N(E + F)\):

We merge the diagrams for \(N(E)\) and \(N(F)\) into one:

\[ N(E) \]
\[ N(F) \]
\[ N(E + F) \]

I.e. given

\[ N(E) = (Q_E, \Sigma, \delta_E, S_E, F_E) \]
\[ N(F) = (Q_F, \Sigma, \delta_F, S_F, F_F) \]

Now we use the disjoint union operation on sets (see the MCS lecture...
In this diagram I only depicted one initial and one final state of each of the automata although they may be several of them.

Here is how we construct $N(\mathcal{E} \mathcal{F})$ from $N(\mathcal{E})$ and $N(\mathcal{F})$:

- The states of $N(\mathcal{E} \mathcal{F})$ are the disjoint union of the states of $N(\mathcal{E})$ and $N(\mathcal{F})$:
  \[ Q_{\mathcal{E} \mathcal{F}} = Q_{\mathcal{E}} + Q_{\mathcal{F}} \]
- The transition function of $N(\mathcal{E} \mathcal{F})$ contains all the transitions of $N(\mathcal{E})$ and $N(\mathcal{F})$ (as for $N(\mathcal{E} + \mathcal{F})$) and for each state $q$ of $N(\mathcal{E})$ which has a transition to a final state of $N(\mathcal{E})$ we add a transition with the same label to all the initial states of $N(\mathcal{F})$.

\[
\delta_{\mathcal{E} \mathcal{F}}((0, q).x) = \left\{ \begin{array}{ll}
0, q' & | q' \in \delta_{\mathcal{E}}(q, x) \\
1, q'' & | q'' \in \delta_{\mathcal{F}}(q, x) \land q'' \in S_{\mathcal{F}}
\end{array} \right.
\]

- The initial states of $N(\mathcal{E} \mathcal{F})$ are the initial states of $N(\mathcal{E})$, and the initial states of $N(\mathcal{E} \mathcal{F})$ if there is an initial state of $N(\mathcal{F})$ which is also final.

\[
S_{\mathcal{E} \mathcal{F}} = \{ 0, q \in S_{\mathcal{E}} \} \cup \{ 1, q \in S_{\mathcal{F}} \}
\]

- The final states of $N(\mathcal{E} \mathcal{F})$ are the final states of $N(\mathcal{E})$.

\[
F_{\mathcal{E} \mathcal{F}} = \{ 1, q \in S_{\mathcal{F}} \}
\]

We now set

\[
N(\mathcal{E} \mathcal{F}) = (Q_{\mathcal{E} \mathcal{F}}, \Sigma, \delta_{\mathcal{E} \mathcal{F}}, S_{\mathcal{E} \mathcal{F}}, F_{\mathcal{E} \mathcal{F}})
\]

I hope that you are able to convince yourself that

\[
L(N(\mathcal{E} \mathcal{F})) = \{ \text{uv} | u \in L(N(\mathcal{E})) \land v \in L(N(\mathcal{F})) \}
\]

and hence we can reason

\[
L(N(\mathcal{E} \mathcal{F})) = \{ \text{uv} | u \in L(N(\mathcal{E})) \land v \in L(N(\mathcal{F})) \} = L(\mathcal{E} \mathcal{F})
\]

6. $N(\mathcal{E}^*)$

We construct $N(\mathcal{E}^*)$ from $N(\mathcal{E})$ by merging initial and final states of $N(\mathcal{E})$ in a way similar to the previous construction and we add a new state $*$ which is initial and final.
Given

\[ N(E) = (Q_E, \Sigma, \delta_E, S_E, F_E) \]

we construct \( N(E^*) \).

- We add one extra state \( * \):

  \[ Q_{E^*} = Q_E + \{ * \} \]

- \( N_{E^*} \) inherits all transitions form \( N_E \) and for each state which has an arrow to the final state labelled \( x \) we add an arrow to all the initial states labelled \( x \).

  \[ \delta_{E^*}((0, q), x) := (0, q') \quad | q' \in \delta_E(q, x) \]
  \[ \quad \cup \{ (0, q') \mid \exists q' \in \delta_E(q, x) \land F_E \neq \emptyset \land q' \in S_E \} \]

- The initial states of \( N(E^*) \) are the initial states of \( N(E) \) and \( * \):

  \[ S_{E^*} = \{ (0, q) \mid q \in S_E \} \cup \{ (1, *) \} \]

- The final states of \( N_{E^*} \) are the final states of \( N_E \) and \( * \):

  \[ F_{E^*} = \{ (0, q) \mid q \in F_E \} \cup \{ (1, *) \} \]

We define

\[ N(E^*) = (Q_{E^*}, \Sigma, \delta_{E^*}, S_{E^*}, F_{E^*}) \]

We claim that

\[ L(N(E^*)) = \{ w_0w_1 \ldots w_{n-1} \mid n \in \mathbb{N} \land \forall i < n \cdot w_i \in L(N(E)) \} \]

since we can run through the automaton an arbitrary number of times. The new state \( * \) allows us also to accept the empty sequence. Hence:

\[ L(N(E^*)) = L(N(E))^* \]

7. \( N(E^*) = N(E) \)

I.e. using brackets does not change anything.

As an example we construct \( N(a^*b^*) \). First we construct \( N(a) \):

Now we have to apply the \( * \)-construction and we obtain:

\[ N(a^*) \]

\[ N(b^*) \]

is just the same and we get
and now we have to serialize the two automata and we get:

Now, you may observe that this automaton, though correct, is unnecessary complicated, since we could have just used

However, we shall not be concerned with minimality at the moment.

### 3.4 Summing up . . .

From the previous section we know that a language given by regular expression is also recognized by a NFA. What about the other way: Can a language recognized by a finite automaton (DFA or NFA) also be described by a regular expression?

The answer is yes:

**Theorem 3.2 (Theorem 3.4, page 91)** Given a DFA $A$ there is a regular expression $R(A)$ which recognizes the same language $L(A) = L(R(A))$.

We omit the proof (which can be found in the [HIMU01] on pp.91-93). However, we conclude:

**Corollary 3.3** Given a language $L \subseteq \Sigma^*$ the following is equivalent: