3 Regular expressions

Given an alphabet Σ 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 nondeterministic 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 expresion) 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 Σ (e.g. $\Sigma = \{\mathrm{a}, \mathrm{b}, \mathrm{c}, \ldots, \mathrm{z}\}$) and define the *syntax* of regular expressions (over Σ)

1. $\emptyset$ is a regular expression.
2. ϵ is a regular expression.
3. For each $x \in \Sigma$, $\mathbf{x}$ 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 $\mathbf{a}$.
4. If E and F are regular expressions then $E + F$ is a regular expression.
5. If E and F are regular expressions then EF (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:

- ϵ
- hallo
- hallo + hello

- h(a + e)llo
- $\mathrm{a}^*\mathrm{b}^*$
- $(\epsilon + \mathrm{b})(\mathrm{ab})^*(\epsilon + \mathrm{a})$

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

- $*$ binds stronger then sequence and +. E.g. we read ab^* as $\mathrm{a}(\mathrm{b}^*)$. We have to use parentheses to enforce the other reading $(\mathrm{ab})^*$.
- 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.

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

$$L^* = \{w_0 w_1 \ldots w_{n-1} \mid n \in \mathbb{N} \wedge \forall i < n. w_i \in L\}$$

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 $L = \{\mathrm{a}, \mathrm{ab}\} \subseteq \{\mathrm{a}, \mathrm{b}\}^*$:

$$L^* = \{\epsilon, \mathrm{a}, \mathrm{ab}, \mathrm{aab}, \mathrm{aba}, \mathrm{aaab}, \mathrm{aaba}, \ldots\}$$

You should notice that we use the same symbol as in Σ^* but there is a subtle difference: Σ 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:

$$w \in L^* \wedge v \in L^* \implies wv \in L^*$$

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

1. $L(\emptyset) = \emptyset$
2. $L(\epsilon) = \{\epsilon\}$
3. $L(\mathbf{x}) = \{x\}$
   where $x \in \Sigma$.
4. $L(E + F) = L(E) \cup L(F)$
5. $L(EF) = \{wv \mid w \in L(E) \wedge v \in L(F)\}$
6. $L(E^*) = L(E)^*$
7. $L((E)) = L(E)$

Subtle points: in 1. the symbol $\emptyset$ may be used as a regular expression (as in $L(\emptyset)$) or the empty set ($\emptyset = \{\}$). Similarily, ϵ 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 langauges they define:

ϵ

$$L(\epsilon) = \{\epsilon\}$$

By 2.

hallo

Let's just look at $L(\text{ha})$. We know from 3:

$$L(\text{h}) = \{\text{h}\}$$
$$L(\text{a}) = \{\text{a}\}$$

Hence by 5:

$$\begin{aligned} L(\text{ha}) &= \{wv \mid w \in \{\text{h}\} \wedge v \in \{\text{a}\}\} \\ &= \{\text{ha}\} \end{aligned}$$

Continuing the same reasoning we obtain:

$$L(\text{hallo}) = \{\text{hallo}\}$$

hallo + hello

From the previous point we know that:

$$L(\text{hallo}) = \{\text{hallo}\}$$
$$L(\text{hello}) = \{\text{hello}\}$$

Hence by using 4 we get:

$$\begin{aligned} L(\text{hallo} + \text{hello}) &= \{\text{hallo}\} \cup \{\text{hello}\}\} \\ &= \{\text{hallo}, \text{hello}\} \end{aligned}$$

h(a + e)llo

Using 3 and 4 we know

$$L(\text{a} + \text{e}) = \{\text{a}, \text{e}\}$$

Hence using 5 we obtain:

$$\begin{aligned} L(\text{h(a} + \text{e)llo}) &= \{uvw \mid u \in L(\text{h}) \wedge v \in L(\text{a} + \text{e}) \wedge w \in L(\text{llo})\} \\ &= \{uvw \mid u \in \{\text{h}\} \wedge v \in \{\text{a}, \text{e}\} \wedge w \in \{(llo\}\} \\ &= \{\text{hallo}, \text{hello}\} \end{aligned}$$

[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:

$$w^i = \underbrace{ww \ldots w}_{i \text{ times}}$$

Now using 6 we know that

$$\begin{aligned} L(\text{a}^*) &= \{w_0 w_1 \ldots w_{n-1} \mid n \in \mathbb{N} \wedge \forall i < n . w_i \in L(\text{a})\} \\ &= \{w_0 w_1 \ldots w_{n-1} \mid n \in \mathbb{N} \wedge \forall i < n . w_i \in \{\text{a}\}\} \\ &= \{\text{a}^n \mid n \in \mathbb{N}\} \end{aligned}$$

and hence using 5 we conclude

$$\begin{aligned} L(\text{a}^*\text{b}^*) &= \{uv \mid u \in L(\text{a}^*) \wedge v \in L(\text{b}^*)\} \\ &= \{uv \mid u \in \{\text{a}^n \mid n \in \mathbb{N}\} \wedge v \in \{\text{a}^m \mid m \in \mathbb{N}\}\} \\ &= \{\text{a}^n\text{b}^m \mid m, n \in \mathbb{N}\} \end{aligned}$$

I.e. $L(\text{a}^*\text{b}^*)$ is the set of all words which start with a (possibly empty) sequence of **a**s followed by a (possibly empty) sequence of **b**s.

$(\epsilon + \mathbf{b})(\mathbf{ab})^*(\epsilon + \mathbf{a})$

Let's analyze the parts:

$$L(\epsilon + \text{b}) = \{\epsilon, \text{b}\}$$
$$L((\text{ab})^*) = \{\text{ab}^i \mid i \in \mathbb{N}\}$$
$$L(\epsilon + \text{b}) = \{\epsilon, \text{b}\}$$

Hence, we have

$$L((\epsilon + \text{b})(\text{ab})^*(\epsilon + \text{a})) = \{u(\text{ab})^i v \mid u \in \{\epsilon, \text{b}\} \wedge i \in \mathbb{N} \wedge v \in \{\epsilon, \text{b}\}$$

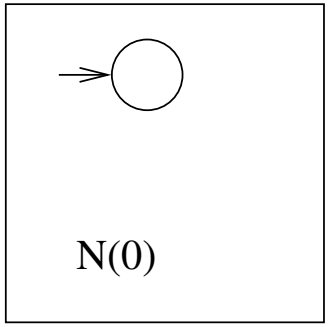
In english: $L((\epsilon + \text{b})(\text{ab})^*(\epsilon + \text{a}))$ is the set of (possibly empty) sequences of interchanging **a**s and **b**s.

3.3 Translating regular expressions to NFAs

Theorem 3.1 *For each regular expression E we can construct ab 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:

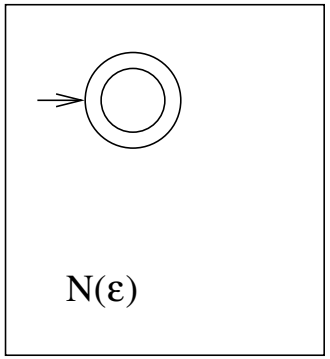
1. $N(\emptyset)$:



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

$$\begin{aligned} L(N(\emptyset)) &= \emptyset \\ &= L(\emptyset) \end{aligned}$$

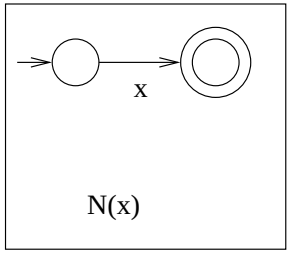
2. $N(\epsilon)$:



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

$$\begin{aligned} L(N(\epsilon)) &= \{\epsilon\} \\ &= L(\epsilon) \end{aligned}$$

3. $N(\mathbf{x})$:

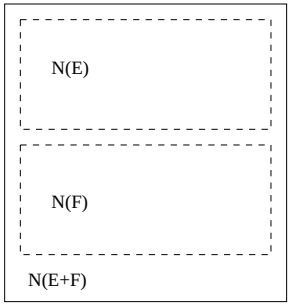


This automaton only accepts the word x, hence:

$$\begin{aligned} L(N(\mathbf{x})) &= \{\mathrm{x}\} \\ &= L(\mathbf{x}) \end{aligned}$$

4. $N(E+F)$:

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



I.e. given

$$\begin{aligned} N(E) &= (Q_E, \Sigma, \delta_E, S_E, F_E) \\ N(F) &= (Q_F, \Sigma, \delta_F, S_F, F_F) \end{aligned}$$

Now we use the disjoint union operation on sets (see the MCS lecture

notes [Alt01], section 4.1)

$$
\begin{aligned}
Q_{E+F} &= Q_E + Q_F \\
\delta_{E+F}((0,q),x) &= \{(0,q') \mid q' \in \delta_E(q,x)\} \\
\delta_{E+F}((1,q)),x &= \{(1,q') \mid q' \in \delta_F(q,x)\} \\
S_{E+F} &= S_E + S_F \\
F_{E+F} &= F_E + F_F
\end{aligned}
$$

$$N(E+F) = (Q_{E+F}, \Sigma, \delta_{E+F}, S_{E+F}, F_{E+F})$$

The disjoint union just signals that we are not going to identify states, even if they accidently happen to have the same name.

Just thinking of the game with markers you should be able to convince yourself that

$$L(N(E+F)) = L(N(E)) \cup L(N(F))$$

Moreover to show that

$$L(N(E+F)) = L(E+F)$$

we are allowed to assume that

$$
\begin{aligned}
L(N(E)) &= L(E) \\
L(N(F)) &= L(F)
\end{aligned}
$$

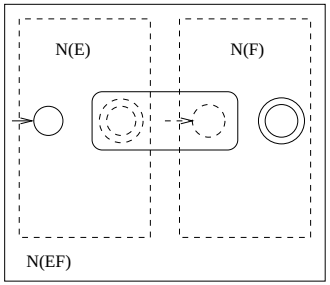
that's what is meant by *induction over the syntax of regular expressions.*

Now putting everything together:

$$
\begin{aligned}
L(N(E+F)) &= L(N(E)) \cup L(N(F)) \\
&= L(E) \cup L(F) \\
&= L(E+F)
\end{aligned}
$$

5. $N(EF)$:

   We want to put the two automata $N(E)$ and $N(F)$ in series. We do this by *connecting* the final states of $N(E)$ with the initial states of $N(F)$ in a way explained below.



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 $N(EF)$ from $N(E)$ and $N(F)$:

$$
\begin{aligned}
N(E) &= (Q_E, \Sigma, \delta_E, S_E, F_E) \\
N(F) &= (Q_F, \Sigma, \delta_F, S_F, F_F)
\end{aligned}
$$

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

$$
\begin{aligned}
\delta_{EF}((0,q),x) &= \{(0,q') \mid q' \in \delta_E(q,x)\} \\
&\quad \cup \{(1,q'') \mid \exists q'.q' \in \delta_E(q,x) \wedge q'' \in S_E\} \\
\delta_{EF}((1,q)) &= \{(1,q') \mid q' \in \delta_F(q))\}
\end{aligned}
$$

- The initial states of $N(EF)$ are the initial states of $N(E)$, and the initial states of $N(F)$ **if** there is an initial state of $N(E)$ which is also a final state.
$$S_{EF} = \{(0,q) \mid q \in S_E\} \cup \{(1,q) \mid q \in S_F \wedge S_E \cap F_E \neq \emptyset\}$$
- The final states of $N(EF)$ are the final states of $N(F)$.
$$F_{EF} = \{(1,q) \mid q \in F_F\}$$

We now set

$$N(EF) = (Q_{EF}, \Sigma, \delta_{EF}, S_{EF}, Z_{EF})$$

I hope that you are able to convince yourself that

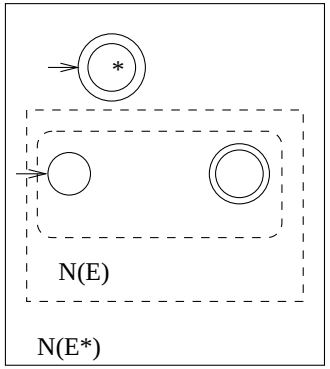
$$L(N(EF)) = \{uv \mid u \in L(N(E)) \wedge v \in L(N(F))$$

and hence we can reason

$$
\begin{aligned}
L(N(EF)) = \{uv \mid u \in L(N(E)) \wedge v \in L(N(F))\} &= \{uv \mid u \in L(E) \wedge v \in L(F) \\
&= L(EF)
\end{aligned}
$$

6. $N(E^*)$:

   We construct $N(E^*)$ from $N(E)$ by merging initial and final states of $N(E)$ 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 also add an arrow to all the initial states labelled x.

$$\begin{aligned}\delta_{E^*}((0,q),x) =& \{(0,q') \mid q' \in \delta_E(q,x)\} \\ & \cup \{(0,q') \mid \exists \delta_E(q,x) \cap F_E \neq \emptyset \wedge q' \in S_E\}\end{aligned}$$

- 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

$$L(N(E^*)) = \{w_0 w_1 \dots w_{n-1} \mid n \in \mathbb{N} \wedge \forall i < n.w_i \in L(N(E))\}$$

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

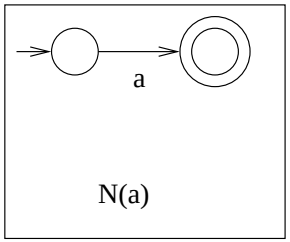
$$\begin{aligned}L(N(E^*)) &= \{w_0 w_1 \dots w_{n-1} \mid n \in \mathbb{N} \wedge \forall i < n.w_i \in L(N(E))\} \\ &= L(N(E))^* \\ &= L(E)^* \\ &= L(E^*)\end{aligned}$$

7. $N((E)) = N(E)$

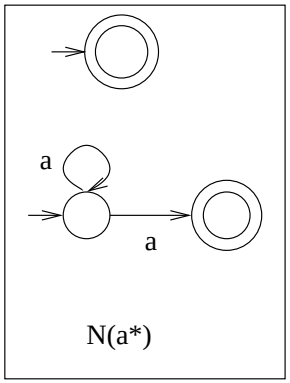
   I.e. using brackets does not change anything.

□

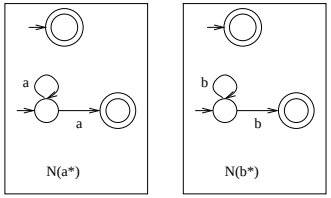
As an example we construct $N(\mathrm{a}^*\mathrm{b}^*)$. First we construct $N(\mathbf{a})$:



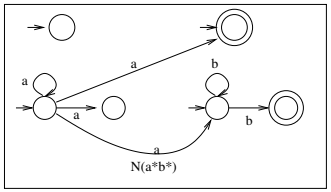
Now we have to apply the $*$-construction and we obtain:



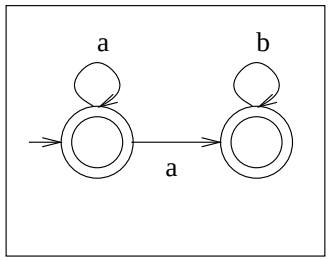
$N(b^*)$ is just the same and we get



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 [HMU01] on pp.91-93). However, we conclude:

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

1. *L is given by a regular expression.*
2. *L is the language accepted by an NFA.*
3. *L is the language acceped by a DFA.*

Proof: We have that 1. $\Longrightarrow$ 2 by theorem 3.1. We know that 2. $\Longrightarrow$ 3. by2.2 and 3. $\Longrightarrow$ 1. by 3.2. □

As indicated in the introduction: the languages which are characterized by any of the three equivalent conditions are called *regular languages* or *type-3-languages*.