Regular expressions ctd.
Formal languages

NFA for $(0+1)^*1(0+1)$

See blackboard.

Exercises

Convert each of the following regular expressions to an ϵ-NFA:

1. 01^*
2. $(0+1)01$
3. $00(0+1)^*$

The big picture (part 1/2)

- Trivially, every DFA is an NFA, and every NFA is an ϵ-NFA.
- One goal is to show that all three types of automata accept the same languages.
- To show this, it suffices to show that for every ϵ-NFA there is a DFA that accepts the same language.
- To that end, we shall use a **modified powerset construction.**

The big picture (part 2/2)

- We have also seen that every regular expression is accepted by an ϵ-NFA.
- We shall see later that for every FA there is a regular expression describing the same language.
- So all **four** formalisms (DFA's, NFA's, ϵ-NFA's, and regular expressions) describe the same languages.

From ϵ-NFA to DFA

- Suppose that N is an ϵ-NFA. We shall now study the **modified powerset construction**, which produces a DFA D that accepts the same language as N.
- To that end, we need one auxiliary definition: given a set S of states of N, the **ϵ-closure** $cl(S)$ of S is the set of states that are reachable from S by any number of ϵ-transitions.

From ϵ-NFA to DFA

The construction of D from N looks as the powerset construction, except that we use cl:

- The alphabet of D is that of N.
- The states of D are sets of the form $cl(S)$, where $S \in P(N)$.
- The initial state q_0^D of D is $cl\{q_0^N\}$.
- The final states of D are those sets of the form $cl(S)$ that contain a final state of N:

$$F_D = \{cl(S) \,|\, S \cap F_N \neq \emptyset\}$$

From ϵ-NFA to DFA

- The transition function of D arises from the transition function of N as follows:

$$\delta_D(S, a) = \bigcup_{q \in S} cl(\delta_N(q, a))$$

That is, $\delta_D(S, a)$ is the set of all states of N that are reachable from some state $q \in S$ via a, followed by any number of ϵ-transitions.

The simulation proposition

Proposition. For every ϵ-NFA N, the DFA D resulting from the modified powerset construction accepts the same language.

Proof. One shows that every string w that

$$w \in L(D) \iff w \in L(N).$$

The proof works by induction on the length of w, and is only slighty more complicated than the proof we have seen for the (ordinary) powerset construction.

Exercise

Consider the following ϵ-NFA.

	ϵ	a	b	c
$\rightarrow p$	$\emptyset$	$\{p\}$	$\{q\}$	$\{r\}$
q	$\{p\}$	$\{q\}$	$\{r\}$	$\emptyset$
$*r$	$\{q\}$	$\{r\}$	$\emptyset$	$\{p\}$

1. Compute the ϵ-closure of each state.
2. Give all strings of length three or less accepted by this automaton.
3. Convert the automaton to a DFA.

Exercise

Repeat the previous exercise for the following ϵ-NFA.

	ϵ	a	b	c
$\rightarrow p$	$\{q, r\}$	$\emptyset$	$\{q\}$	$\{r\}$
q	$\emptyset$	$\{p\}$	$\{r\}$	$\{p, q\}$
$*r$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

Exercise

In an earlier exercise, we converted the regular expressions 01^*, $(0+1)01$, and $00(0+1)^*$ into ϵ-NFA's. Convert each of those ϵ-NFA's into a DFA.

Formal languages

Formal languages: overview

- A **formal language** (or simpy “language”) is a set L of strings over some finite alphabet Σ. That is, a subset $L \subseteq \Sigma^*$.
- Finite automata and regular expressions describe certain formal languages.
- But many important formal languages, e.g. programming languages, are not regular.
- To describe describe formal languages, we shall use **formal grammars**.

Formal grammars: overview

- Important for describing programming languages.
- Different kinds of formal grammars are described by the the **Chomsky hierarchy**.
- Among the simplest grammars in the Chomsky hierarchy are the **regular grammars**, which—as we shall see—describe the same languages as regular expressions.

Formal grammars: basic idea

To generate strings by beginning with a **start symbol** S and then apply rules that indicate how certain combinations of symbols may be replaced with other combinations of symbols.

Formal grammar: definition

Definition. A formal grammar $G = (N, \Sigma, P, S)$ consists of

- a finite set N of **non-terminal symbols**;
- a finite set Σ of **terminal symbols** not in N;
- a finite set P of **production rules** of the form

$$u \longrightarrow v$$

  where u and v are strings in $(\Sigma \cup N)^*$ and u contains at least one non-terminal symbol;
- a **start symbol** S in N.

Example

The grammar G with non-terminal symbols $N = \{S\}$, terminal symbols $\Sigma = \{a, b\}$, and productions

$$S \to aSb$$
$$S \to ab.$$

Following a common practice, we use capital letters for non-terminal symbols and small letters for terminal symbols.

Example

Arithmetic expressions (simplified). $N = \{E, I\}$, $\Sigma = \{a, b, 0, 1, *, +, (,)\}$, $S = E$, and P as below:

$$
\begin{array}{ll}
E \longrightarrow I & I \longrightarrow a \\
E \longrightarrow E + E & I \longrightarrow b \\
E \longrightarrow E * E & I \longrightarrow Ia \\
E \longrightarrow (E) & I \longrightarrow Ib \\
 & I \longrightarrow I0 \\
 & I \longrightarrow I1
\end{array}
$$

Example: compact notation

More compact notation for productions:

$$E \rightarrow I \mid E + E \mid E * E \mid (E)$$

$$I \rightarrow a \mid b \mid Ia \mid Ib \mid I0 \mid I1$$

Think of E as “expressions” and I as “identifiers”.

Language of a formal grammar

Definition. The language of a formal grammar $G = (N, \Sigma, P, S)$, denoted as $L(G)$, is defined as all those strings over Σ that can be generated by starting with the start symbol S and then applying the production rules in P until no more non-terminal symbols are present.

Exercises

The language of all palindromes over the alphabet $\{a, b, c\}$. (A **palindrome** is a word of the form vw such that w is the reverse of v, e.g. "abba".)