2.6 REGULAR EXPRESSIONS

A different way of describing simple languages.

Let A and B be languages over Σ. Basic operations:

- *Union:* $A \cup B = \{x \in \Sigma^* \mid x \in A \text{ or } x \in B\}$;
- *Catenation:* $AB = \{xy \in \Sigma^* \mid x \in A,\ y \in B\}$;
- *Powers:*

$$\begin{cases} A^0 &= \{\varepsilon\}, \\ A^k &= AA^{k-1} = \{x_1 \dots x_k \mid x_i \in A \quad \forall i = 1, \dots, k\} \qquad (k \geq 1); \end{cases}$$

- *Closure or "Kleene star":*

$$\begin{aligned} A^* &= \bigcup_{k \geq 0} A^k \\ &= \{x_1 \dots x_k \mid k \geq 0,\ x_i \in A \quad \forall i = 1, \dots, k\}. \end{aligned}$$

Definition 2.3 The *regular expressions* over the alphabet Σ are inductively defined by

1. $\emptyset$ ja ε are regular expressions over Σ;
2. $\boldsymbol{a}$ is a regular expression over Σ for all $a \in \Sigma$;
3. if r and s are regular expressions over Σ, then $(r \cup s)$, (rs) and r^* are regular expressions over Σ;
4. no other regular expressions over Σ exist.

Each regular expression r over Σ defines a regular language $L(r)$ defined by:

- $L(\emptyset) = \emptyset$;
- $L(\varepsilon) = \{\varepsilon\}$;
- $L(\boldsymbol{a}) = \{a\}$ for all $a \in \Sigma$;
- $L((r \cup s)) = L(r) \cup L(s)$;
- $L((rs)) = L(r)L(s)$;
- $L(r^*) = (L(r))^*$.

Regular expressions over $\{a, b\}$:

$$r_1 = \boldsymbol{((ab)b)}, \quad r_2 = \boldsymbol{(ab)^*},$$

$$r_3 = \boldsymbol{(ab^*)}, \quad r_4 = \boldsymbol{(a(b \cup (bb)))^*}.$$

Languages described by the regular expressions:

$$\begin{aligned} L(r_1) &= (\{a\}\{b\})\{b\} = \{ab\}\{b\} = \{abb\}; \\ L(r_2) &= \{ab\}^* = \{\varepsilon, ab, abab, ababab, \dots\} \\ &= \{(ab)^i \mid i \geq 0\}; \\ L(r_3) &= \{a\}(\{b\})^* = \{a, ab, abb, abbb, \dots\} \\ &= \{ab^i \mid i \geq 0\}; \\ L(r_4) &= (\{a\}\{b, bb\})^* = \{ab, abb\}^* \\ &= \{\varepsilon, ab, abb, abab, ababb, \dots\} \\ &= \{x \in \{a, b\}^* \mid \text{ every } a\text{-symbol in } x \\ &\qquad \text{is followed by 1 or 2 } b\text{-symbols}\}. \end{aligned}$$

Rules for reducing parenthesis:

Operator precedence:

$$* \quad \succ \quad \cdot \quad \succ \quad \cup$$

Associativity of union and catenation:

$$\begin{aligned} L(((r{\cup}s){\cup}t)) &= L((r{\cup}(s{\cup}t))) \\ L(((rs)t)) &= L((r(st))) \end{aligned}$$

⇒ consecutive unions and catenations need not be parenthesized.

Regular letters are used if no danger of confusion with strings exists.

More simply, then:

$$r_1 = abb, \quad r_2 = (ab)^*, \quad r_3 = ab^*, \quad r_4 = (a(b \cup bb))^*.$$

Definition 2.4 A language is *regular*, if it can be described by a regular expression.

Simplifying regular expressions

Regular languages usually have alternative expressions, e.g.:

$$\begin{aligned} \Sigma^* &= L((a \cup b)^*) \\ &= L((a^*b^*)^*) \\ &= L(a^*b^* \cup (a \cup b)^*ba(a \cup b)^*). \end{aligned}$$

Definition. Regular expressions r and s are *equivalent*, denoted by $r = s$, if $L(r) = L(s)$.

Simplifying an expression = determining the “simplest” equivalent expression.

Equivalence testing of regular expressions is a nontrivial, but in theory mechanically solvable problem.

Rules for simplification:

$$\begin{aligned} r \cup (s \cup t) &= (r \cup s) \cup t \\ r(st) &= (rs)t \\ r \cup s &= s \cup r \\ r(s \cup t) &= rs \cup rt \\ (r \cup s)t &= rt \cup st \\ r \cup r &= r \end{aligned} \qquad \begin{aligned} r \cup \emptyset &= r \\ \varepsilon r &= r \\ \emptyset r &= \emptyset \\ r^* &= \varepsilon \cup r^*r \\ r^* &= (\varepsilon \cup r)^* \end{aligned}$$

Any true equivalence of regular expression can be derived from these rules and the rule

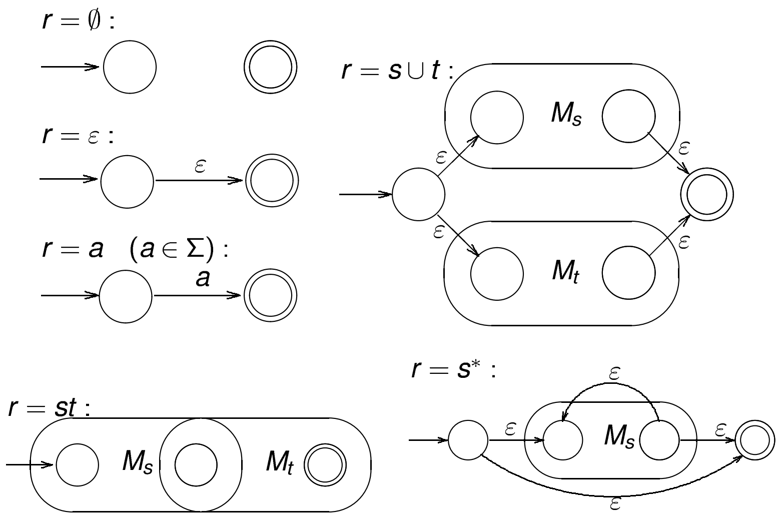
if $r = rs \cup t$, then $r = ts^*$, given that $\varepsilon \notin L(s)$.

2.7 FINITE STATE AUTOMATA AND REGULAR LANGUAGES

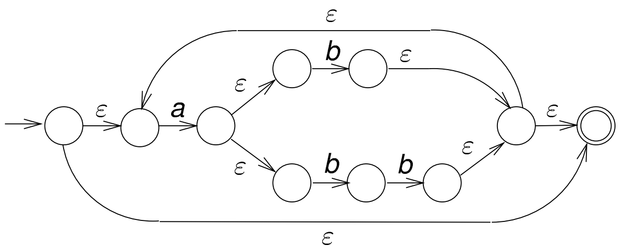
Theorem 2.3 Every regular language can be recognized by a finite state automaton.

Proof. By the following inductive construction one may form an ε-automaton that follows the structure of a given regular expression. For this automaton, $L(M_r) = L(r)$. If necessary, the ε-transitions can be removed as per Lemma 2.4 and the automaton may be determinized by the construction in Theorem 2.2.

One should note that in the construction the ε-automata always have a unique initial and final state, and there is no transition internal to the subautomaton that would leave from the final state or enter into the initial state. ☐

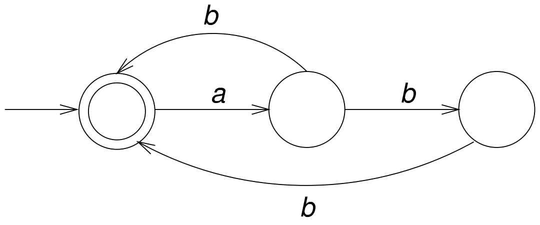

$r = \emptyset$:
$r = \varepsilon$:
ε
$r = a \quad (a \in \Sigma)$:
a
$r = s \cup t$:
M_s
M_t
ε
ε
ε
ε
$r = st$:
M_s
M_t
$r = s^*$:
ε
ε
M_s
ε
ε

For example, from $r = (a(b \cup bb))^*$ we may obtain the following ε-automaton:


ε
ε
a
ε
b
ε
ε
b
b
ε
ε
ε

The automaton is clearly very redundant. When designing automata by hand it is often good to design them directly.

For example it is easy to design the simple nondeterministic automaton to recognize the language that corresponds to the regular expression $r = (a(b \cup bb))^*$:



Lause 2.4 Every language that can be recognized by a finite state automaton is regular.

Proof. One more extension of finite state automata is needed: a *generalized nondeterministic finite state automaton* (GNFA) allows the use of regular expressions as conditions for a transition.

Formalization: Let RE_Σ = the set of regular expressions over Σ. A *GNFA* is a 5-tuple

$$M = (Q, \Sigma, \delta, q_0, F),$$

where the transition function δ is a finite mapping

$$\delta : Q \times \mathrm{RE}_\Sigma \to \mathcal{P}(Q)$$

(i.e., $\delta(q, r) \neq \emptyset$ only for finitely many pairs $(q, r) \in Q \times \mathrm{RE}_\Sigma$).

The single step successor relation is defined:

$$(q, w) \underset{M}{\vdash} (q', w')$$

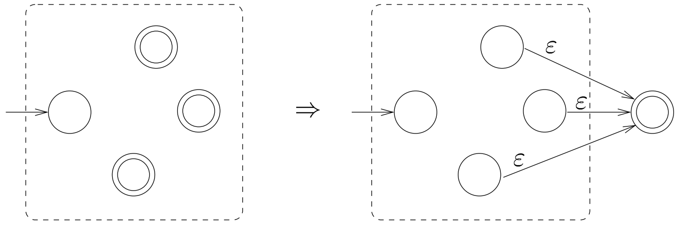
if $q' \in \delta(q, r)$ for some $r \in \mathrm{RE}_\Sigma$, such that $w = zw'$, $z \in L(r)$. Other definitions as previously.

We prove: every language recognized by a GNFA is regular.

Let M be a GNFA. The regular expression that describes the language M recognizes is computed in two stages:

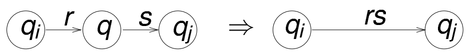
1. The automaton M is condensed to an equivalent GNFA with at most two states by the following transformations:

(i) If M has more than one final state, they are combined as follows.

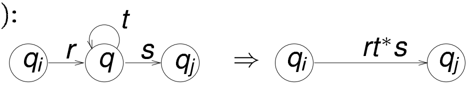


(ii) States other than the initial and final state are removed one by one as follows. Let q be a state of M, that is neither the initial nor the final state; examine all “paths” that pass via q in M. Let q_i and q_j be the immediate predecessor and successor of q on some such path. We remove q from the path $q_i \to q_j$ by the transformation (i), if there is no transition from q to itself, or by transformation (ii), if there is a transition from q to itself:

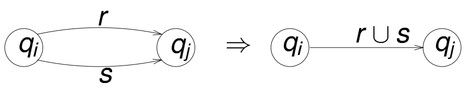
(i):



(ii):

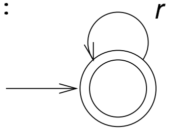


Simultaneously, parallel transitions are joined as follows:



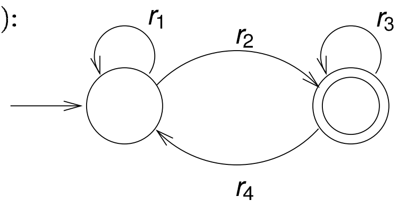
2. At the end of the reduction the regular expression that corresponds to the remaining automaton with at most 2 states is formed as in the picture:

(i):



$\Rightarrow \quad r^*$

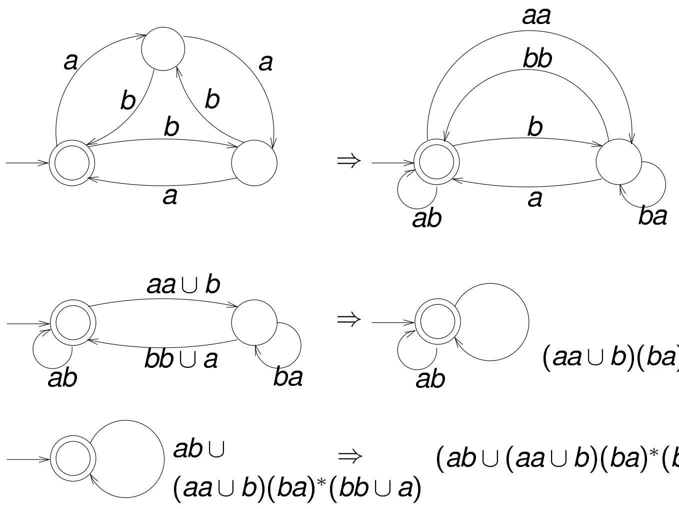
(ii):



$\Rightarrow \quad r_1^* r_2 (r_3 \cup r_4 r_1^* r_2)^*$

□

Example:


a
a
b
b
b
a
$\Rightarrow$
aa
bb
b
a
ab
ba
$\Rightarrow$
$aa \cup b$
$bb \cup a$
ab
ba
$\Rightarrow$
ab
$\Rightarrow$
$ab \cup$
$(aa \cup b)(ba)^*(bb \cup a)$
$\Rightarrow$

$(aa \cup b)(ba)^*(bb \cup a)$

$(ab \cup (aa \cup b)(ba)^*(bb \cup a))^*$