CSE2315 Slides week 2 - Regular Expressions

Matthijs Spaan   Stefan Hugtenburg

Algorithmics group

Delft University of Technology

19 February 2020

This lecture

- Regular expressions and their formal definition
- Equivalence of regular expressions and regular languages
- Constructing an NFA from a regular expression
- Generalized nondeterministic finite automata (GNFAs)
- Generalized transition graphs
- Acceptation and recognition by GNFAs
- Constructing a regular expression from an NFA

Special languages and operations on languages

Special languages: $\boxed{\emptyset}$, $\boxed{\{\varepsilon\}}$, $\boxed{\{a\}}$, Σ and Σ^*.

Boolean operations: $\overline{L}$, $\boxed{L_1 \cup L_2}$, $L_1 \cap L_2$, $L_1 - L_2$

$$\begin{array}{rcl}
L^R & = & \{w^R \mid w \in L\} \\
\boxed{L_1 L_2} & = & \{vw \mid v \in L_1 \text{ and } w \in L_2\} \\
L^2 & = & LL \\
L^3 & = & LLL \\
\boxed{L^*} & = & L^0 \cup L^1 \cup \cdots \cup L^n \cup \cdots \\
L^+ & = & L^1 \cup L^2 \cup \cdots \cup L^n \cup \cdots
\end{array}$$

Regular expressions

(see also video Regular Expressions)

These are expressions/formulas with which regular languages can be described.

To construct them, we use $\emptyset$, ε, symbols from the alphabet, and the operations

$$\cup \qquad \circ \qquad {}^*$$

along with parentheses '(' and ')'. For instance:

$$a \circ b \circ (c \cup d)^*.$$

Note: Usually we write $ab(c \cup d)^*$.
In the literature, $ab(c + d)^*$ is also often used.

Also compare with the framed formulas on the previous slide.

Regular expressions

Definition: Let Σ be a given alphabet.

1. Then $\emptyset$, ε and a, for each $a \in \Sigma$, are primitive regular expressions.
2. If R_1 and R_2 are regular expressions, then $(R_1 \cup R_2)$, $(R_1 \circ R_2)$ and (R_1^*) are also regular expressions.
3. Only expressions that can be obtained using a finite number of applications of 1. and 2. are regular expressions.

Abbreviations: $R^+ = R \circ R^*$, $R_1R_2 = R_1 \circ R_2$ and $\Sigma = a_0 \cup \cdots \cup a_n$, when $\Sigma = \{a_0, \ldots, a_n\}$.
We will be liberal about the use of parentheses, with * binding stronger than $\circ$, which in turn binds stronger than $\cup$. For instance: $abc^* \cup ab$ stands for $(((a \circ b) \circ (c^*)) \cup (a \circ b))$.

Set $\mathscr{R}$ of regular expressions

Just as in logic the set of propositions can be recursively (inductively) defined, we can define the set $\mathscr{R}$ of all regular expressions.

Definition: The set $\mathscr{R}$ of all regular expressions over a given alphabet Σ is the smallest set that satisfies:

1. $a, \varepsilon, \emptyset \in \mathscr{R}$ for all $a \in \Sigma$.
2. If $R_1, R_2 \in \mathscr{R}$, then $(R_1 \cup R_2), (R_1 \circ R_2), (R_1^*) \in \mathscr{R}$.

NB: This definition is almost the same as that on the previous slide, which defines what regular expressions are. Here we define the set of regular expressions.

Interpretation regular expressions

$L(R)$ stands for the language that is described by the regular expression R and is recursively (inductively) defined as:

$$\begin{aligned}
L(\emptyset) &= \emptyset \\
L(\varepsilon) &= \{\varepsilon\} \\
L(a) &= \{a\}, \quad \text{for each } a \in \Sigma \\
L(R_1 \cup R_2) &= L(R_1) \cup L(R_2) \\
L(R_1 \circ R_2) &= L(R_1)L(R_2) \\
L(R_1^*) &= L(R_1)^*
\end{aligned}$$

Examples

$$
\begin{aligned}
L((a \cup b)c) &= \{ac, bc\} \\
L((a \cup b)(c \cup \varepsilon)) &= \{a, b, ac, bc\} \\
L((a \cup b)^* b (a \cup b)^* b (a \cup b)^*) &= \\
&\{w \in \{a,b\}^* \mid w \text{ contains at least two } b\text{s}\} \\
L(c^*(b \cup ac^*)^*) &= \\
&\{w \in \{a,b,c\}^* \mid w \text{ contains no substring } bc\}
\end{aligned}
$$

Exercise

Which language does the following regular expression represent?

$$a \circ a \circ a^* \circ (\varepsilon \cup b \cup (b \circ b))$$

or written differently:

$$aaa^*(\varepsilon \cup b \cup bb)$$

Solution

The language L containing all words with two or more as followed by at most two bs, i.e.,

$$L = \{a^m b^n \mid m \geq 2 \text{ and } n \leq 2\}.$$

Exercise

Give a regular expression R such that:

$$L(R) = \{w \in \{0,1\}^* \mid w \text{ contains the substring } 00\}$$

Equivalence regular expressions and regular languages

(see also video Equivalence of Regular Expressions and Regular Languages)

Theorem: Let L be a language over some alphabet Σ. Then:

L regular iff there exists a regular expressions R such that $L(R) = L$.

The proof consists of two parts:

1. For every regular expression R, we construct an NFA M such that $L(M) = L(R)$.
2. For every NFA M we construct a regular expression R such that $L(R) = L(M)$.

Regular expressions and regular languages

Theorem: If R is a regular expression, then there exists an NFA M such that $L(M) = L(R)$. As a consequence of this, $L(R)$ is a regular language.

Proof: Using induction on R, construct an NFA $M(R)$ such that $L(M(R)) = L(R)$. The constructed NFA has exactly one accept state.

Basis step: Construct $M(\emptyset)$, $M(\varepsilon)$ and $M(a)$, for each $a \in \Sigma$.

Inductive step: Given NFAs $M(R_1)$ and $M(R_2)$, construct $M(R_1 \cup R_2)$, $M(R_1 \circ R_2)$ and $M(R^*)$.

For the (simple) constructions, see Sipser (Lemma 1.55) or the video mentioned on the previous slide.

Exercise

Let $R = ((ab)^* \cup (ba^*))^*$.

Construct an NFA N such that $L(N) = L(R)$.

Regular languages en regular expressions

Theorem: If L is a regular language, then there exists a regular expression R such that $L = L(R)$.

Proof idea: Using generalized nondeterministic finite automata (GNFAs), in which the transitions are labeled with regular expressions over Σ instead of using symbols from Σ_ε.
The finite automaton recognizing L is transformed into a GNFA in which subsequently all but the starting and accepting state are eliminated one by one. While doing this, the regular expressions on the transitions are modified such that the new GNFA recognizes the same language. Ultimately, the transition from the starting to the accepting state is labeled with the desired regular expression.

GNFAs: Informally

A GNFA is a special NFA $(Q, \Sigma, \delta, q_s, q_a)$, where:

1. the transitions are labeled with regular expressions over Σ;
2. there exists exactly one start state q_a;
3. there exist no transitions to the start state q_s;
4. there exist no transitions from the accept state q_a;
5. between every pair of nodes $q_i, q_j \in Q$ there exists a transition $\delta(q_i, q_j) \in \mathscr{R}$ such that $i \neq a$ and $j \neq s$ (because of 3 and 4 above). Here, $\mathscr{R}$ is the set of all regular expressions over Σ.

NB 1: Every NFA can be transformed into a(n) (G)NFA that satisfies all requirements above. See also the previous lecture (with regards to 2–4).

NB 2: Transitions can be labeled by $\emptyset$.

Definition GNFA

Definition: A Gegeneralized nondeterministic finite automaton (GNFA) is a 5-tuple $(Q, \Sigma, \delta, q_s, q_a)$, where:

1. Q is a finite set of states;
2. Σ is the input alphabet;
3. $\delta : (Q - \{q_a\}) \times (Q - \{q_s\}) \to \mathscr{R}$ is the transition function;
4. q_s is the start state; and
5. q_a is the accept state.

Acceptation by GNFAs: Informally

A generalized transition graph is a transition graph belonging to a GNFA. In a generalized transition graph, edges labeled with $\emptyset$ are usually left out. In the following, we do not discern a GNFA and its generalized transition graph.

A GNFA accepts a word w if in the generalized transition graph a path exists from the start state to the accepting state where the edges of the path are labeled with a sequence of regular expressions $R_1, \ldots, R_k$, such that $w \in L(R_1 \circ \ldots \circ R_k)$.

An NFA (transition graph) and a GNFA (generalized transition graph) are equivalent if they accept the same words.

Definition acceptation by GNFA

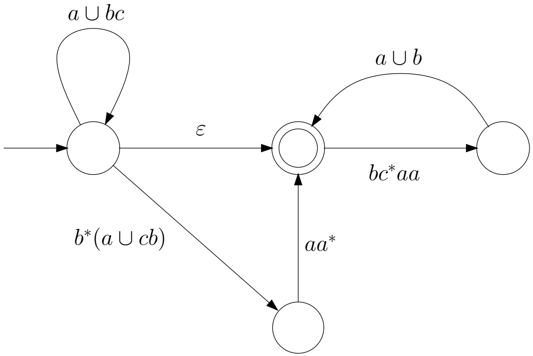
Definition: A GNFA $(Q, \Sigma, \delta, q_s, q_a)$ accepts a word $w \in \Sigma^*$ if w can be written as $w = w_1 w_2 \dots w_k$ with $w_1, w_2, \dots, w_k \in \Sigma^*$ and there exists a sequence of states $q_0, q_1, \dots q_k \in Q$ such that:

1. $q_0 = q_s$,
2. $q_k = q_a$, and
3. for all i $(1 \leq i \leq k)$: $w_i \in L(R_i)$ where $R_i = \delta(q_{i-1}, q_i)$.

NB: In the third requirement above, $\delta(q_{i-1}, q_i)$ represents the regular expression R_i corresponding to the transition from state q_{i-1} to q_i. Otherwise formulated, this requirement states that:

$$w = w_1 \dots w_k \in L(R_1) \circ \dots \circ L(R_k) = L(R_1 \circ \dots \circ R_k).$$

Example and exercise



Which of the following strings $w \in \{a, b, c\}^*$ are accepted?
(a) $abcabbaa$ (b) $cbbb$ (c) $bccaab$

Answers

(a) yes

(b) no

(c) yes

Language recognized by a GNFA

Definition: If M is a GNFA, then the language $L(M)$ recognized by M is defined as:

$$L(M) = \{w \in \Sigma^* \mid M \text{ accepts } w\}.$$

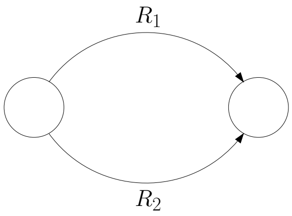
From NFA to GNFA

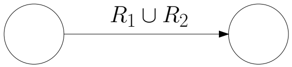
An NFA M can be transformed in a straightforward manner to a GNFA M' that recognizes the same language as M. The algorithm that performs this task is:

1. If necessary, create a new start state without incoming transitions (see previous lecture).
2. If necessary, create a new unique accept state without outgoing transitions.
3. Merge parallel transitions (next slide).
4. If necessary, add transitions with label $\emptyset$ (slide 25).

Merging transitions

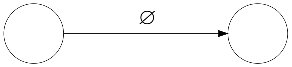
If in an NFA two states are connected in the same direction by two (or more) transitions labeled by regular expressions R_1 and R_2 (first automaton), then these can be replaced by one transition with label $R_1 \cup R_2$ in the GNFA (second automaton):



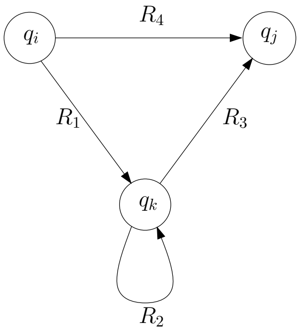


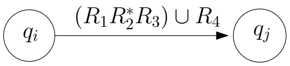
Unconnected states

If two states in a GNFA are not connected (first automaton), then one can connect them with the regular expression $\emptyset$ (second automaton):



Eliminating a state (q_k)

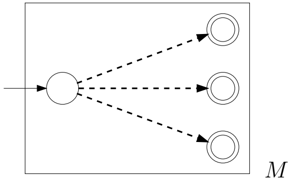


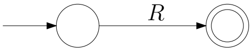


Note: For all q_i and q_j (and in reverse!) around q_k the same recipe must be followed. Do not forget the case where $q_i = q_j$.
Also: $L(\emptyset^*) = \emptyset^* = \{\varepsilon\}$ (relevant when $R_2 = \emptyset$).

Construction continued

- Every NFA M can be straightforwardly transformed into an equivalent GNFA.
- For every GNFA (with more than two nodes) there exists an equivalent GNFA with one node less (but other and more complex regular expressions as labels).
- Every NFA M can therefore be transformed into a GNFA with two nodes and one transition that is labeled with a regular expression R such that $L(R) = L(M)$.



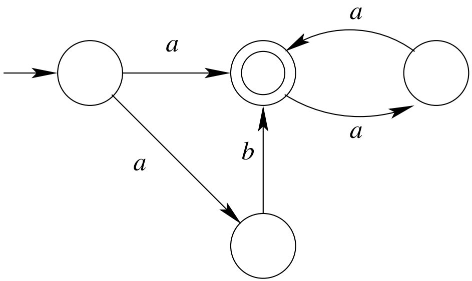


Structure inductive proof of correctness construction

Sipser defines a recursive function $\mathrm{CONVERT}(G)$ that transforms a GNFA G into a regular expression. This function calls itself recursively on a GNFA in which one node has been removed in accordance with the elimination procedure. The result of $\mathrm{CONVERT}(G')$ for a GNFA G' with two nodes is the regular expression on the transition from the starting state to the accept state.

Sipser uses induction over the number of nodes in G to prove that for all GNFAs G, the result R of $\mathrm{CONVERT}(G)$ satisfies $L(R) = L(G)$.

Exercise



Construct a regular expression $L(M)$ that the above NFA M accepts.

Solution

$$(a \cup ab)(aa)^*$$