Einführung in die Computerlinguistik

Reguläre Ausdrücke und reguläre Grammatiken

Laura Kallmeyer

Heinrich-Heine-Universität Düsseldorf

Summer 2018

Regular expressions (1)

Let Σ be an alphabet. The set of regular expressions over Σ is recursively defined:

- $\emptyset$ is a regular expression denoting the set $\emptyset$.
- ϵ is a regular expression denoting the set $\{\epsilon\}$.
- For each $a \in \Sigma$: a is a regular expression denoting $\{a\}$.
- If r and s are regular expressions denoting R and S, then $(r|s)$, (rs) and (r^*) are also regular expressions denoting $R \cup S$, $R \circ S$ and R^* respectively.

Assume that $*$ has a higher priority than concatenation which in turn has a higher priority than $|$.
a^+ is an abbreviation for aa^*.

Regular expressions (2)

Bsp.:

- $\varepsilon|a$ denotiert $\{\varepsilon\} \cup \{a\} = \{a, \varepsilon\}$.
- εa denotiert $\{\varepsilon\} \circ \{a\} = \{a\}$.
- $\emptyset|a$ denotiert $\emptyset \cup \{a\} = \{a\}$.
- $\emptyset a$ denotiert $\emptyset \circ \{a\} = \emptyset$.
- cc^* denotiert die Menge aller Wörter über $\{c\}$, deren Länge ≥ 1 ist.
- $(cc)^*$ denotiert die Menge aller Wörter über $\{c\}$, deren Länge eine gerade Zahl ist.

Regular expressions (3)

- $(a|\varepsilon)bcc^*$ denotiert $\{bc, abc, bcc, abcc, bccc, abccc, \dots\} = \{a^n bc^m \mid 0 \leq n \leq 1, m \geq 1\}$
- $((a|b)^*|de)^*$ denotiert $\{w_1 \dots w_n \mid n \geq 0$, jedes der w_i für $1 \leq i \leq n$ ist entweder aus $\{a, b\}^*$ oder hat die Form $(de)^k$ für irgendein $k \geq 0\}$
  $\Rightarrow$ die Sprache ist die Menge aller Wörter über den Alphabet $\{a, b, d, e\}$, in denen jedes d notwendig von einem e gefolgt ist.
  Äquivalent: $(a|b|de)^*$
- $a(bd^+b)^*a$ denotiert die Menge aller Wörter der Form awa, wobei $w = \varepsilon$ oder $w = bw'b$ und es gilt, dass $w' \in \{b, d\}^*$ mit d anfängt und endet und dass in w' jede Gruppe von bs genau die Länge 2 hat.

Regular expressions (4)

The following holds:
For each language L: there is a regular expression x with $L = L(x)$ iff there is a FSA M with $L = L(M)$.

In order to show this, we

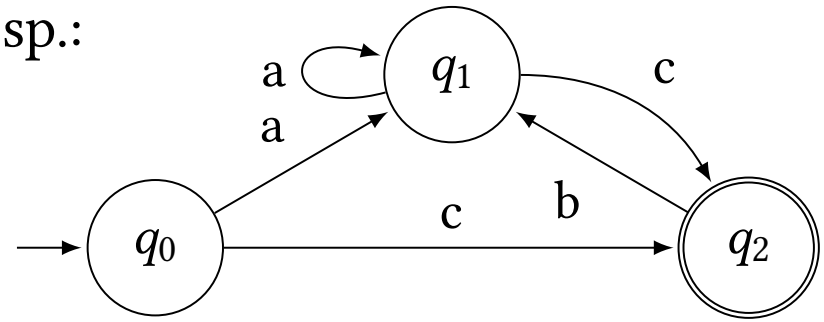
1. define NFAs with empty transitions and show their equivalence to NFAs;
2. show how to construct an equivalent NFA with empty transitions for a given regular expression;
3. show how to construct an equivalent regular expression for a given DFA.

Zu 1. und 2. siehe Vorlesung “Mathematische Grundlagen der Computerlinguistik”.

DFAs and regular expressions (1)

For each DFA, there is an equivalent regular expression.

Bsp.:



Um von q_0 zu q_2 zu gelangen, kann man entweder

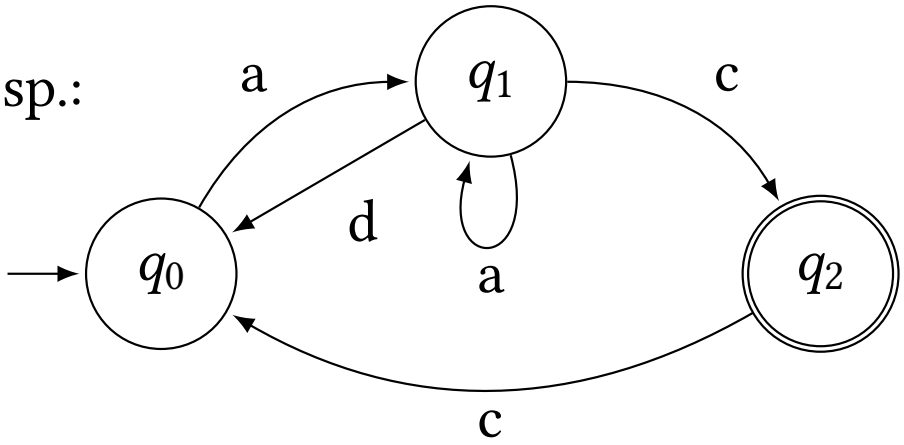
(1) einen Weg ohne Unterwegszustand q_2 wählen. Dies kann wiederum entweder ohne Durchlauf von q_1 sein (c) oder ein erstes Mal von q_0 zu q_1 (a), dann beliebig oft von q_1 zu q_1 ohne Durchlauf von q_1 (a^*), dann von q_1 nach q_2 (c) $\Rightarrow$ $(c|a^+c)$, oder

(2) man geht ein erstes Mal von q_0 zu q_2 $(c|a^+c)$ und dann beliebig oft von q_2 zu q_2 ohne Durchlauf von q_2 $((ba^*c)^*)$.

$\Rightarrow (c|a^+c)(ba^*c)^*$

DFAs and regular expressions (2)

Bsp.:



von q_0 nach q_2 ohne q_1, q_2	$\emptyset$
von q_0 nach q_1 ohne q_1, q_2	a
von q_1 nach q_1 ohne q_1, q_2	$a\|da$
von q_1 nach q_2 ohne q_1, q_2	c
von q_0 nach q_2 ohne q_2	$\emptyset\|a(a\|da)^*c = a(a\|da)^*c$
von q_2 nach q_2 ohne q_2	$ca(a\|da)^*c$
von q_0 nach q_2	$a(a\|da)^*c(ca(a\|da)^*c)^*$

DFAs and regular expressions (3)

Algorithm for construction the regular expression for a given DFA:
Assume that $\{q_1, \ldots, q_n\}$ it the set of states.
Define $R^k_{i,j}$ as the set of sequences of input symbols that can be obtained from following all paths from q_i to q_j that do not pass through states q_l with $l > k$.
Question: What do the $R^k_{i,j}$ look like and what are the corresponding regular expressions $r^k_{i,j}$?

- $k = 0$: Only paths of length 0 or 1 can be considered, since, between q_i and q_j, no other states q_l are possible.
  Each $r^0_{i,j}$ has the form $a_1|a_2|\cdots|a_m$ or (if $i = j$) $a_1|a_2|\cdots|a_m|\epsilon$ or (if $i \neq j$ and there is no edge from q_i to q_j) $\emptyset$.

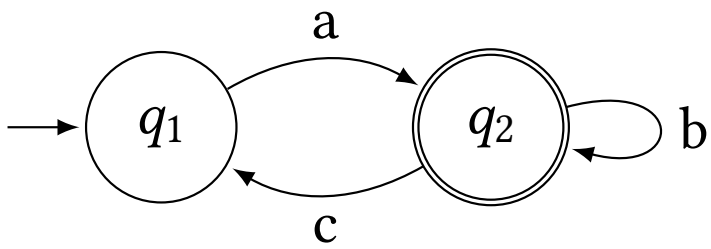
DFAs and regular expressions (4)

- In general, R_{ij}^k contains
  - all elements from R_{ij}^{k-1},
  - all concatenations of
    a) an element from R_{ik}^{k-1},
    b) any number (including 0) of elements from R_{kk}^{k-1} and
    c) an element from R_{kj}^{k-1}.

  Consequently, $r_{i,j}^k = r_{ij}^{k-1} | r_{ik}^{k-1} (r_{kk}^{k-1})^* r_{kj}^{k-1}$.

Finally, the regular expression for the language accepted by the automaton is $r_{1f_1}^n | r_{1f_2}^n | \cdots | r_{1f_m}^n$ if $F = \{q_{f_1}, \ldots, q_{f_m}\}$.

DFAs and regular expressions (5)



$$r_{12}^2 = r_{12}^1 | r_{12}^1 (r_{22}^1)^* r_{22}^1$$

$$\begin{aligned} r_{12}^1 &= r_{12}^0 | r_{11}^0 (r_{11}^0)^* r_{12}^0 \\ &= a | \epsilon (\epsilon^*) a = a \end{aligned}$$

$$\begin{aligned} r_{22}^1 &= r_{22}^0 | r_{21}^0 (r_{11}^0)^* r_{12}^0 \\ &= (b | \epsilon) | c (\epsilon)^* a = \epsilon | b | ca \end{aligned}$$

$$r_{12}^2 = a | a(\epsilon | b | ca)^+ = a(\epsilon | b | ca)^* = a(b | ca)^*$$

Abschlusseigenschaften regulärer Sprachen (1)

Die von regulären Ausdrücken denotierten Sprachen heißen reguläre Sprachen.

1. Wenn L_1 und L_2 reguläre Sprachen sind, dann
   - ist die Vereinigung von L_1 und L_2 ($L_1 \cup L_2$) ebenfalls eine reguläre Sprache.
   - ist die Schnittmenge von L_1 und L_2 ($L_1 \cap L_2$) ebenfalls eine reguläre Sprache.
   - ist die Konkatenation von L_1 und L_2 ($L_1 \circ L_2$) ebenfalls eine reguläre Sprache.
2. Das Komplement einer regulären Sprache ist eine reguläre Sprache.
3. Wenn L eine reguläre Sprache, dann ist L^* eine reguläre Sprache.

Abschlusseigenschaften regulärer Sprachen (2)

Für Vereinigung, Konkatenation und L^* kann man ganz einfach die entsprechenden regulären Ausdrücke angeben:

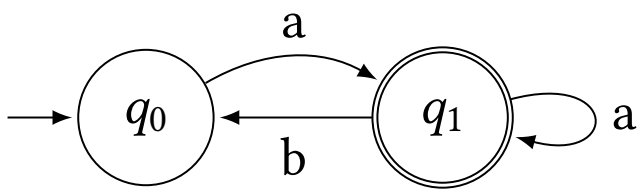
Seien r_1 und r_2 reguläre Ausdrücke mit $L_1 = L(r_1)$ und $L_2 = L(r_2)$. Dann gilt

- $L(r_1|r_2) = L_1 \cup L_2$
- $L(r_1 r_2) = L_1 \circ L_2$
- $L(r_1^*) = L_1^*$

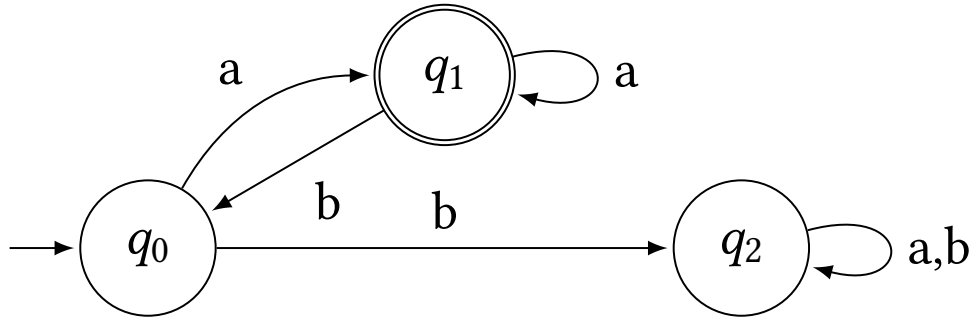
Abschlusseigenschaften regulärer Sprachen (3)

Für die Komplementbildung nimmt man den DFA, der die Sprache erkennt, macht die Übergangsfunktion vollständig, indem man einen weiteren Zustand (*trap state*) hinzufügt, und wählt als neue Endzustände alle Zustände, die im Ursprungsautomaten kein Endzustand sind.

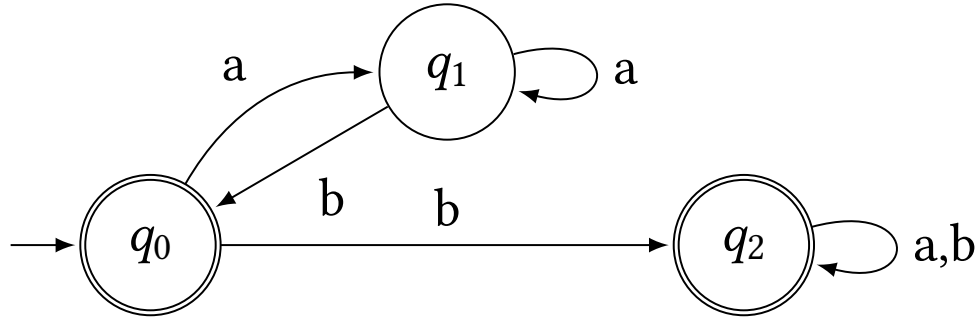
Bsp.:



Trap state:



Komplementbildung:



Abschlusseigenschaften regulärer Sprachen (4)

Schnittbildung kann (de Morgan) durch Vereinigung und Komplementbildung ausgedrückt werden:

$$L_1 \cap L_2 = \overline{\overline{L_1 \cap L_2}} = \overline{\overline{L_1} \cup \overline{L_2}}$$

Damit ist $L_1 \cap L_2$ eine reguläre Sprache, falls L_1 und L_2 regulär sind.

Regular Grammars (1)

Another way to define a language is via a grammar. A grammar formalism defines the form of rules and combination operations allowed in a grammar.

A context-free grammar (CFG) G is a tuple $\langle N, T, P, S \rangle$ with

- N and T disjoint alphabets, the nonterminals and terminals,
- $S \in N$ the start symbol, and
- P a set of productions of the form $A \to \alpha$ with $A \in N, \alpha \in (N \cup T)^*$.

Regular Grammars (2)

Let $G = \langle N, T, P, S \rangle$ be a CFG. The *(string) language* $L(G)$ of G is the set $\{w \in T^* \mid S \overset{*}{\Rightarrow} w\}$ where

- for $w, w' \in (N \cup T)^*$: $w \Rightarrow w'$ iff there is a $A \to \alpha \in P$ and there are $v, u \in (N \cup T)^*$ such that $w = vAu$ and $w' = v\alpha u$.
- $\overset{*}{\Rightarrow}$ is the reflexive transitive closure of $\Rightarrow$:
  - $w \overset{0}{\Rightarrow} w$ for all $w \in (N \cup T)^*$, and
  - for all $w, w' \in (N \cup T)^*$: $w \overset{n}{\Rightarrow} w'$ iff there is a v such that $w \Rightarrow v$ and $v \overset{n-1}{\Rightarrow} w'$.
  - for all $w, w' \in (N \cup T)^*$: $w \overset{*}{\Rightarrow} w'$ iff there is a $i \in \mathbb{N}$ such that $w \overset{i}{\Rightarrow} w'$.

A language is called a context-free language (CFL) iff it is generated by a CFG.

Regular Grammars (3)

A CFG is called

- right-linear iff for all productions $A \to \beta$: $\beta \in T^*$ or $\beta = \beta' X$ with $\beta' \in T^*, X \in N$.
- left-linear iff for all productions $A \to \beta$: $\beta \in T^*$ or $\beta = X\beta'$ with $\beta' \in T^*, X \in N$.
- regular if it is left-linear or right-linear.

For each left-linear grammar there is an equivalent right-linear grammar and vice versa.

Regular Grammars (4)

Some examples: A CFG $G = \langle \{S, A, B\}, \{a, b, c\}, P, S \rangle$

1. with $P = \{S \to abS \mid cA, A \to bB, B \to cbB \mid \varepsilon\}$ is right-linear.
   It generates the language $L((ab)^* cb(cb)^*)$.
2. with $P = \{S \to Sa \mid Sb \mid Abc, A \to B, B \to Bcb \mid \varepsilon\}$ is left-linear.
   It generates the language $L((cb)^* bc(a|b)^*)$.
3. with $P = \{S \to abSA \mid c, A \to b\}$ is not regular.
   It generates the language $\{(ab)^n cb^n \mid n \geq 0\}$.

Regular Grammars (5)

The following holds:
$L = L(G)$ for a regular grammar G iff L is a regular set (i.e., can be described by a regular expression).

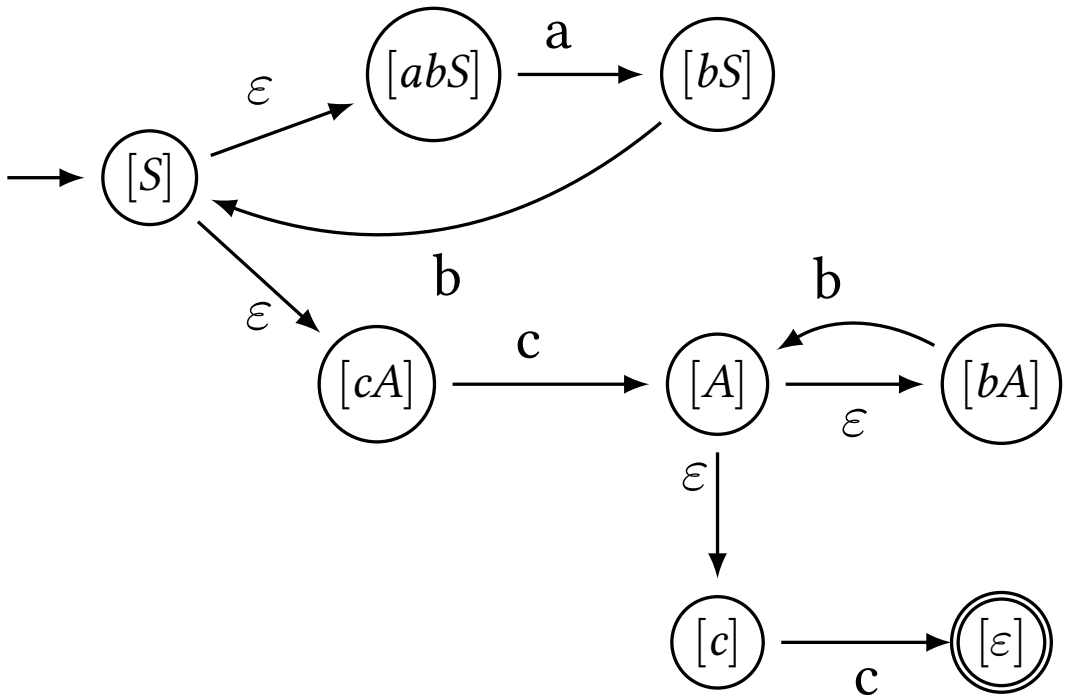
To show this, we show that

- for each right-linear grammar, there exists an equivalent NFA; and
- for each DFA, there is an equivalent right-linear grammar.

Regular Grammars (6)

Example: NFA with empty transitions for a given right-linear grammar.
$G = \langle \{S, A\}, \{a, b, c\}, P, S \rangle$ with $P = \{S \to abS \mid cA, A \to bA \mid c\}$

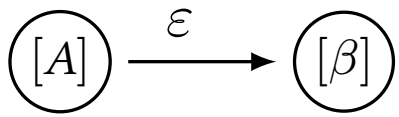
Corresponding NFA:

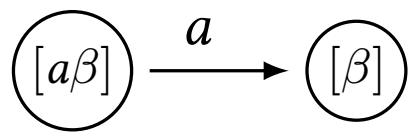


Regular Grammars (7)

Construction of equivalent NFA (with empty transitions) for given right-linear G:

- Start state $[S]$.
- For all productions $A \to \beta_1\beta_2$ there is a state $[\beta_2]$.
- Transitions simulate top-down left-to-right traversal of derivation tree:

  For every $A \to \beta$: 

  For every state $[a\beta]$ with $a \in T$: 

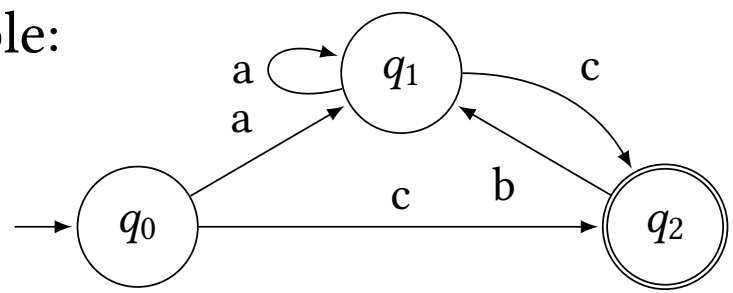
- Final state is $[\epsilon]$.

Regular Grammars (8)

Construction of a right-linear grammar for given DFA:

- The states are the nonterminals.
- The start state is the start symbol.
- For each transition $Q \xrightarrow{a} Q'$ add a production $Q \to aQ'$ and, if $Q' \in F$, a production $Q \to a$.
- If the start state Q_0 is a final state, then add $Q_0 \to \varepsilon$

Example:



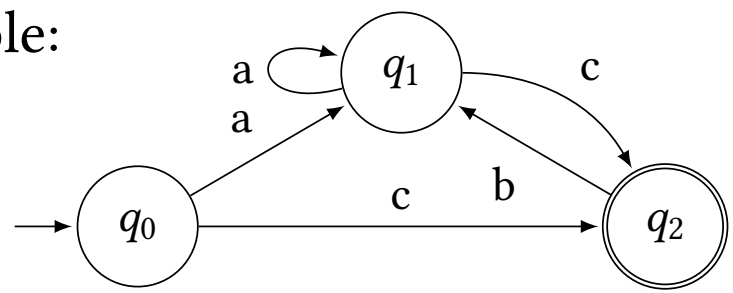
$$\langle \{q_0, q_1, q_2\}, \{a, b, c\},$$
$$\{q_0 \to aq_1 \mid cq_2 \mid c,$$
$$q_1 \to aq_1 \mid cq_2 \mid c, q_2 \to bq_1\},$$
$$q_0 \rangle$$

Regular Grammars (9)

For a left-linear grammar, we have to follow the transitions in the opposite direction.

- The nonterminals are $Q \cup \{S\}$ where S is a new start symbol.
- $q_0 \to \varepsilon$ where q_0 is the DFA's start state and $S \to q_f$ for all $q_f \in F$.
- For each transition $q \xrightarrow{a} q'$ we add $q' \to qa$.

Example:



$$\langle \{q_0, q_1, q_2, S\}, \{a, b, c\},$$
$$\{S \to q_2, q_0 \to \varepsilon,$$
$$q_2 \to q_1 c \mid q_0 b,$$
$$q_1 \to q_1 a \mid q_0 a \mid q_2 b\},$$
$$S\rangle$$