

Einführung in die Computerlinguistik

Reguläre Ausdrücke und reguläre Grammatiken

Laura Kallmeyer

Heinrich-Heine-Universität Düsseldorf

Summer 2022



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.¹

Assume that $*$ has a higher priority than concatenation which in turn has a higher priority than $|$.

a^+ is an abbreviation for aa^* .

$a?$ is sometimes used as an abbreviation for $a|\epsilon$.

¹ $R \circ S = \{w_1w_2 \mid w_1 \in R, w_2 \in S\}$. $R^* = \{w_1 \dots w_n \mid n \geq 0, w_1, \dots, w_n \in R\}$

Beispiel: reguläre Ausdrücke

- $\varepsilon|a$ 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)

Beispiel: reguläre Ausdrücke

- $(a|\varepsilon)bcc^*$ denotiert $\{bc, abc, bcc, abcc, bccc, abccc, \dots\} = \{a^nbc^m \mid 0 \leq n \leq 1, m \geq 1\}$
- $((a|b)^*|de)^*$ denotiert $\{w_1 \dots w_n \mid n \geq 0, \text{ jedes der } w_i \text{ für } 1 \leq i \leq n \text{ ist entweder aus } \{a, b\}^* \text{ oder hat die Form } (de)^k \text{ für irgendein } k \geq 0\}$
 \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 und jedes e notwendig auf ein d folgt.
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.

Beispiel: reguläre Ausdrücke

- Die Menge $\{w \in \{a, b\}^* \mid |w|_a \geq 2\}$ wird denotiert von $b^*ab^*a(a|b)^*$
Äquivalent: $b^*a(b^*a)^+b^*$
- Die Menge $\{w \in \{a, b\}^* \mid |w| \text{ ist gerade}\}$ wird denotiert von $((a|b)(a|b))^*$
Äquivalent: $(aa|ab|ba|bb)^*$
- Die Menge $\{w \in \{a, b\}^* \mid \text{jedes } b \text{ in } w \text{ folgt auf ein } a\}$ wird denotiert von $(a|ab)^*$.
Äquivalent: $(a(\varepsilon|b))^*$

$|w|$ = Länge von w , $|w|_a$ = Anzahl a 's in w .

Regular expressions (5)

The following holds:

Equivalence of regular expressions and FSA

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

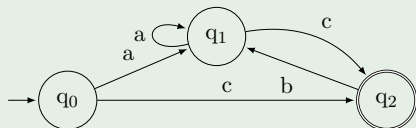
- 1 define NFAs with empty transitions and show their equivalence to NFAs;
- 2 show how to construct an equivalent NFA with empty transitions for a given regular expression;
- 3 show how to construct an equivalent regular expression for a given DFA.

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

DFAs and regular expressions (1)

For each DFA, there is an equivalent regular expression.

Beispiel: reg. Ausdruck für FSA

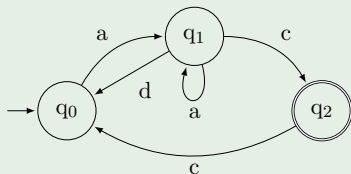


Um von q_0 zu q_2 zu gelangen, kann man entweder

- (1) einen Weg **ohne Unterwgszustand 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)

Beispiel: reg. Ausdruck für FSA

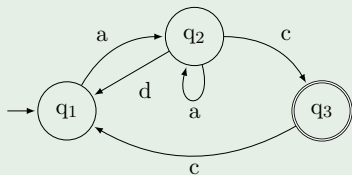


von q_0 nach q_2 ohne q_1, q_2	\emptyset
von q_0 nach q_1 ohne q_1, q_2	a
von q_1 nach q_1 ohne q_1, q_2	$\varepsilon a da$
von q_1 nach q_2 ohne q_1, q_2	c
von q_0 nach q_2 ohne q_2	$\emptyset a(a da)^*c = a(a da)^*c$
von q_2 nach q_2 ohne q_2	$\varepsilon ca(a da)^*c$
von q_0 nach q_2	$a(a da)^*c(ca(a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

DFAs and regular expressions (3)

Algorithm for construction the regular expression for a given DFA: Assume that $\{q_1, \dots, q_n\}$ it the set of states.

- Define $R_{i,j}^k$ 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 $l > k$.
- Define $r_{i,j}^k$ as the corresponding regular expression.



$$r_{1,3}^1 = \emptyset$$

$$r_{2,2}^1 = \varepsilon | a | da$$

$$r_{1,3}^2 = \emptyset | a(a|da)^*c = a(a|da)^*c$$

$$r_{1,3}^3 = a(a|da)^*c(ca(a|da)^*c)^*$$

$$r_{1,2}^1 = a$$

$$r_{2,3}^1 = c$$

$$r_{3,3}^2 = \varepsilon | ca(a|da)^*c$$

DFAs and regular expressions (4)

Question: What do the $R_{i,j}^k$ look like and what are the corresponding regular expressions $r_{i,j}^k$?

- $k = 0$: 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 $r_{i,j}^0$ 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 (5)

- 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} .

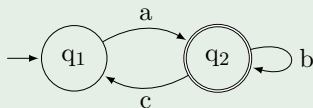
Consequently, $r_{i,j}^k = r_{ij}^{k-1} | r_{ik}^{k-1} (r_{kk}^{k-1})^* r_{kj}^{k-1}$.

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 $F = \{q_{f_1}, \dots, q_{f_m}\}$.

DFAs and regular expressions (6)

wichtig: rekursive Formel $r_{i,j}^k = r_{ij}^{k-1} | r_{ik}^{k-1} (r_{kk}^{k-1})^* r_{kj}^{k-1}$

From DFA to its regular expression



Gesamtausdruck: r_{12}^2 (q2 einziger Endzustand)

$r_{12}^2 = r_{12}^1 | r_{12}^1 (r_{22}^1)^* r_{22}^1$ (rekursive Formel mit $i = 1, j = 2, k = 2$)

$r_{12}^1 = r_{12}^0 | r_{11}^0 (r_{11}^0)^* r_{12}^0$ (rekursive Formel mit $i = 1, j = 2, k = 1$)
 $= a | \epsilon (\epsilon^*) a = a$

$r_{22}^1 = r_{22}^0 | r_{21}^0 (r_{11}^0)^* r_{12}^0$ (rekursive Formel mit $i = 2, j = 2, k = 1$)
 $= (b | \epsilon) | c (\epsilon)^* a = \epsilon | b | ca$

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

Abschlusseigenschaften regulärer Sprachen (1)

Die von regulären Ausdrücken denotierten Sprachen heißen reguläre Sprachen.

Abschlusseigenschaften

- 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:

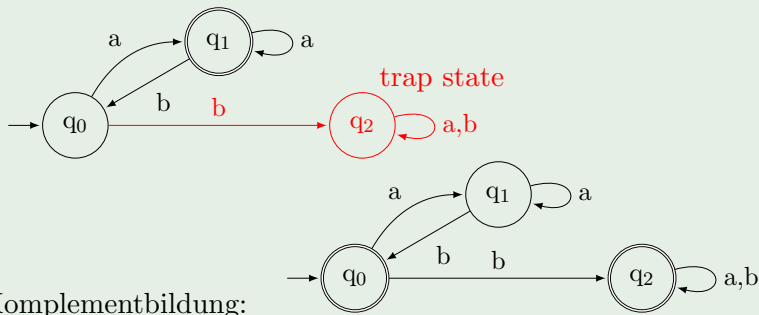
Seien r_1 und r_2 reguläre Ausdrücke mit $L_1 = L(r_1)$ und $L_2 = L(r_2)$. Dann gilt

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

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.

Komplementbildung im DFA



Komplementbildung:

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

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

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 $A \rightarrow \alpha$ 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 $\{w \in T^* \mid S \xRightarrow{*} w\}$ where

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

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 $A \rightarrow \beta$: $\beta \in T^*$ or $\beta = \beta'X$ with $\beta' \in T^*$, $X \in N$.
- **left-linear** iff for all productions $A \rightarrow \beta$: $\beta \in T^*$ 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)

Examples: regular grammars

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

- 1 with $P = \{S \rightarrow abS \mid cA, A \rightarrow bB, B \rightarrow cbB \mid \varepsilon\}$ is right-linear.

It generates the language $L((ab)^*cb(cb)^*)$.

- 2 with $P = \{S \rightarrow Sa \mid Sb \mid Abc, A \rightarrow B, B \rightarrow Bcb \mid \varepsilon\}$ is left-linear.

It generates the language $L((cb)^*bc(a|b)^*)$.

- 3 with $P = \{S \rightarrow abSA \mid c, A \rightarrow b\}$ is not regular.

It generates the language $\{(ab)^n cb^n \mid n \geq 0\}$.

Regular Grammars (5)

Equivalence of regular grammars and regular expressions

$L = L(G)$ 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

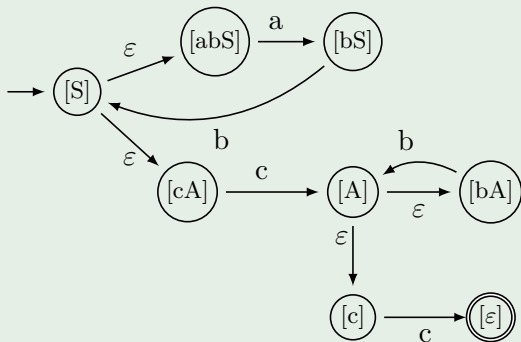
- for each right-linear grammar, there exists an equivalent NFA; and
- for each DFA, there is an equivalent right-linear grammar.

Regular Grammars (6)

Example: Constructing an NFA with empty transition for a right-linear grammar

$G = \langle \{S, A\}, \{a, b, c\}, P, S \rangle$ with $P = \{S \rightarrow abS \mid cA, A \rightarrow bA \mid c\}$

Corresponding NFA:



Regular Grammars (7)

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

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

For every $A \rightarrow \beta$: $\textcircled{[A]} \xrightarrow{\epsilon} \textcircled{[\beta]}$

For every state $[a\beta]$ with $a \in T$: $\textcircled{[a\beta]} \xrightarrow{a} \textcircled{[\beta]}$

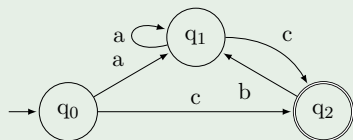
- Final state is $[\epsilon]$.

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 $\textcircled{Q} \xrightarrow{a} \textcircled{Q'}$ add a production $Q \rightarrow aQ'$ and, if $Q' \in F$, a production $Q \rightarrow a$.
- If the start state Q_0 is a final state, then add $Q_0 \rightarrow \varepsilon$

Example: Constructing a right-linear grammar for a DFA



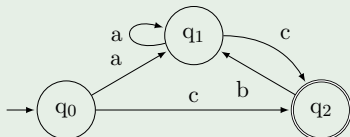
$\langle \{q_0, q_1, q_2\}, \{a, b, c\},$
 $\{q_0 \rightarrow aq_1 \mid cq_2,$
 $q_1 \rightarrow aq_1 \mid cq_2, q_2 \rightarrow bq_1 \mid \varepsilon\},$
 $q_0 \rangle$

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.
- $S \rightarrow q_f$ for all $q_f \in F$.
- For each transition $\textcircled{q} \xrightarrow{a} \textcircled{q'}$ we add $q' \rightarrow qa$.
- $q_0 \rightarrow \varepsilon$ where q_0 is the DFA's start state.

Example: Constructing a left-linear grammar for a DFA


$$\langle \{q_0, q_1, q_2, S\}, \{a, b, c\}, \\ \{S \rightarrow q_2, q_0 \rightarrow \varepsilon, \\ q_2 \rightarrow q_1c \mid q_0c, \\ q_1 \rightarrow q_1a \mid q_0a \mid q_2b\}, \\ S \rangle$$