Algorithmen und Datenstrukturen

Theoretische Grundlagen der Informatik

Prof. Dr. rer. nat. Jörg Striegnitz

Fachhochschule Aachen
striegnitz@fh-aachen.de

29. April 2024

Algorithmen und Datenstrukturen

Reguläre Ausdrücke

Prof. Dr. rer. nat. Jörg Striegnitz

Fachhochschule Aachen
striegnitz@fh-aachen.de

29. April 2024

Reguläre Sprachen / Motivation

Definition 1.57 (Reguläre Sprachen)

Seien Σ ein Alphabet und $L \subseteq \Sigma^*$ eine Sprache. L heißt genau dann **reguläre Sprache**, wenn es einen DEA $M = (Q, \Sigma, \delta, q_0, F)$ mit $L(M) = L$ gibt. Mit $\mathcal{L}_{Reg}$ bezeichnen wir die Menge aller regulären Sprachen.

- **Bisher**: reguläre Sprachen beschrieben durch
  - endliche Automaten, die sie erkennen (DEA,NEA und ε-NEA), und
  - mathematische Mengenschreibweise.
- **Jetzt**: **reguläre Ausdrücke** zur Beschreibung regulärer Sprachen
  - algebraische Beschreibung regulärer Sprachen
  - Definition einer ersten »Programmiersprache« (Syntax und Semantik)
  - Beschreibung eines »Compilers«

Operationen auf Sprachen

- Sprachen sind Mengen, auf die wir die bekannten Mengenoperationen anwenden können. Seien U eine Menge und $A, B \subseteq U$
  - $\cup$ (**Vereinigung**): $A \cup B = \{e \in U \mid e \in A \ \vee \ e \in B\}$
  - $\cap$ (**Schnitt**): $A \cap B = \{e \in U \mid e \in A \ \wedge \ e \in B\}$
  - $-$ (**Differenz**): $A - B = \{e \in U \mid e \in A \ \wedge \ \neg e \in B\}$
  - $\cdot^C$ (**Komplement**): $A^C = \{e \in U \mid \neg e \in A\}$

- Es gelten die bekannten Rechenregeln für Mengen; z.B.
  - **Kommutativgesetz**: $A \circ B = B \circ A$ für $\circ \in \{\cap, \cup\}$
  - **Assoziativgesetz**: $A \circ (B \circ C) = (A \circ B) \circ C$ für $\circ \in \{\cap, \cup\}$
  - **Distributivgesetz**: $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$ bzw. $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$

- Wir wollen weitere Operationen einführen

  u.a. solche, die würdigen, dass Sprachen Mengen von Wörtern sind

Warum interessieren uns Operationen auf Sprachen?

- Weil wir mit Hilfe der zugehörigen Rechenregeln u.U. Probleme vereinfachen können

  Finden von Automaten: **funktionale Dekomposition**

- Beschreibe L ein bestimmtes algorithmisches Problem
  - Gelingt es uns L mithilfe der Rechenregeln z.B. in $L = A \cup (B \cap C)$ zu zerlegen,
  - die »kleineren« Probleme A, B und C zu lösen, um dann aus
  - den Teillösungen die Gesamtlösung systematisch zusammenzusetzen, so
  - können wir das Problem L evtl. einfacher lösen.

Konkatenation von Sprachen

Definition 1.58 (Konkatenation von Sprachen)

Seien L_1 und L_2 Sprachen über Σ, dann ist

$$L_1 \cdot L_2 = L_1 L_2 = \{ vw \mid v \in L_1 \land w \in L_2 \}$$

die **Konkatenation** von L_1 und L_2.

- L_1 liefert Präfixe, L_2 Suffixe.

Beispiel 1.52 (Konkatenation von Sprachen)

Betrachte $L_1 = \{\varepsilon, a, b, abb\}$ und $L_2 = \{0, 1, 010\}$, dann ist

$$L_1 \cdot L_2 = \{0, 1, 010,\ a0, a1, a010,\ b0, b1, b010,\ abb0, abb1, abb010\}$$

Iteration von Sprachen

Definition 1.59 (Iteration von Sprachen)

Sei L eine Sprache, dann definieren wir

$$L^0 = L_\varepsilon = \{\varepsilon\}$$

$$L^{i+1} = L^i \cdot L \quad \forall i \in \mathbb{N}$$

- Ein Wort $w \in L^i$ lässt sich immer in i Wörter aus L zerlegen:

$$w = u_1 \cdot u_2 \cdot \cdots \cdot u_i \ \text{ mit } \ u_i \in L$$

Beispiel: Iteration von Sprachen 1

Beispiel 1.53 (Iteration von Programmiersprachen)

Betrachte $L = \{0, 1\} = \Sigma_{Bool}$

- $L^0 = \{\varepsilon\}$
- $L^1 = L^0 \cdot L = \{\varepsilon\} \cdot \{0, 1\} = \{0, 1\}$
- $L^2 = L^1 \cdot L = \{0, 1\} \cdot \{0, 1\} = \{00, 01, 10, 11\}$
- $\vdots$
- $L^n = L^{n-1} \cdot L = \{w \in \Sigma_{Bool}^* \mid |w| = n\}$

Offenbar ist

$$\bigcup_{n \in \mathbb{N}} \Sigma_{Bool}^n = \Sigma_{Bool}^*$$

Beispiel: Iteration von Sprachen 2

Beispiel 1.54 (Iteration von Programmiersprachen)

Betrachte $L = \{\varepsilon, 0, 1\} = \Sigma_{Bool} \cup \{\varepsilon\}$

- $L^0 = \{\varepsilon\}$
- $L^1 = L^0 \cdot L = \{\varepsilon\} \cdot \{\varepsilon, 0, 1\} = \{\varepsilon, 0, 1\}$
- $L^2 = L^1 \cdot L = \{\varepsilon, 0, 1\} \cdot \{\varepsilon, 0, 1\} = \{\varepsilon, 0, 1, 00, 01, 10, 11\}$
- $\vdots$
- $L^n = L^{n-1} \cdot L = \{w \in \Sigma^*_{Bool} \mid |w| \leq n\}$

Kleene-Stern und Spiegelsprache

Definition 1.60 (Kleene-Stern)

Sei L eine Sprache, dann definieren wir

$$L^* \quad = \quad \bigcup_{n \in \mathbb{N}} L^n$$

als den **Kleene-Stern** von L. Ferner ist $L^+ = L \cdot L^*$.

Bemerkung:

- Offenbar ist $L^+ = L^* - \{\varepsilon\}$, falls $\varepsilon \notin L$.

Definition 1.61 (Spiegelsprache)

Sei L eine Sprache, dann definieren wir

$$L^R \quad = \quad \left\{ w^R \;\middle|\; w \in L \right\}$$

als die **Spiegelsprache** von L.

Syntax regulärer Ausdrücke

Definition 1.62 (Syntax regulärer Ausdrücke)

Sei Σ ein Alphabet. Die Menge der regulären Ausdrücke $\mathcal{R}_\Sigma$ über Σ ist induktiv definiert durch:

1. $\varnothing \in \mathcal{R}_\Sigma$
2. $\varepsilon \in \mathcal{R}_\Sigma$
3. $\Sigma \subseteq \mathcal{R}_\Sigma$
4. falls $r_1, r_2 \in \mathcal{R}_\Sigma$, so auch
   - 4.1 $(r_1 + r_2)$
   - 4.2 $(r_1 \cdot r_2)$
5. falls $r \in \mathcal{R}_\Sigma$, so auch $(r)^*$

Notation:

- Um Klammern einzusparen, legen wir fest:
  - * bindet stärker als $\cdot$,
  - $\cdot$ bindet stärker als $+$.
- Den Punkt darf man weglassen
- Ansonsten benutze Klammern für eindeutige Lesbarkeit

Strukturanalyse von regulären Ausdrücken

Es gibt offenbar mehrere Möglichkeiten einen regulären Ausdruck zu konstruieren:

Ein regulärer Ausdruck, multiple Konstruktionsmöglichkeiten

Seien $\Sigma = \{a, b, c\}$ und $\boxed{r = ((a+b) \cdot (c)^*)}$ Gilt . $r \in \mathcal{R}_\Sigma$?

Variante 1

1. $a \in \mathcal{R}_\Sigma \;\wedge\; b \in \mathcal{R}_\Sigma$ (Regel 3)
2. $(a+b) \in \mathcal{R}_\Sigma$ (1. und Regel 4a)
3. $c \in \mathcal{R}_\Sigma$ (Regel 3)
4. $(c)^* \in \mathcal{R}_\Sigma$ (3. und Regel 5)
5. $((a+b) \cdot (c)^*) \in \mathcal{R}_\Sigma$ (2., 4. und Regel 4b)

Variante 2

1. $c \in \mathcal{R}_\Sigma$ (Regel 3)
2. $(c)^* \in \mathcal{R}_\Sigma$ (1. und Regel 5)
3. $a \in \mathcal{R}_\Sigma \;\wedge\; b \in \mathcal{R}_\Sigma$ (Regel 3)
4. $(a+b) \in \mathcal{R}_\Sigma$ (1. und Regel 4a)
5. $((a+b) \cdot (c)^*) \in \mathcal{R}_\Sigma$ (2., 4. und Regel 4b)

Beachte: Zuletzt angewandte Regel ist **eindeutig**! (hier 4b)

Beispiel: Syntax regulärer Ausdrücke 1

- Betrachte $\Sigma = \{a, b, c\}$ und $r = \left(\left((a)^* + b\right) \cdot c\right)$. Gilt $r \in \mathcal{R}_\Sigma$?
- r kann nur durch Anwendung von Regel 4b) $(\cdot)$ entstanden sein: Konstruktor $\cdot$ wurde auf zwei reguläre Ausdrücke r_1 und r_2 angewendet:

$$\left(\underbrace{\left((a)^* + b\right)}_{r_1} \cdot \underbrace{c}_{r_2}\right)$$

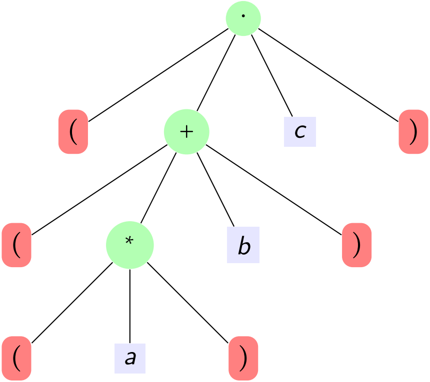
- r ist gültiger regulärer Ausdruck, genau dann, wenn r_1 und r_2 gültige reguläre Ausdrücke.
- Insgesamt ergibt sich folgender **Beweisbaum**:

<table>
<tr><td colspan="3">$r = \left(\left((a)^* + b\right) \cdot c\right)$</td></tr>
<tr><td colspan="3">$\downarrow$ R 4b $(\cdot)$</td></tr>
<tr><td colspan="2">$r_1 = \left((a)^* + b\right)$</td><td>$r_2 = c$ gilt (R 3)</td></tr>
<tr><td colspan="2">$\downarrow$ R 4a $(+)$</td><td></td></tr>
<tr><td>$r_{1.1} = (a)^*$</td><td>$r_{1.2} = b$ gilt (R 3)</td><td></td></tr>
<tr><td>$\downarrow$ R 5 $(^*)$</td><td></td><td></td></tr>
<tr><td>$T_{1.1.1} = a$ gilt (R 3)</td><td></td><td></td></tr>
</table>

- Beweisbaum definiert **eindeutige** Baumstruktur für regulären Ausdruck (siehe nächste Folie).

Breispiel: Syntax regulärer Ausdrücke 2

- Baumdarstellung zu $r = \left(\left((a)^* + b\right) \cdot c\right)$:



- Man spricht vom ***concrete syntax Tree*** oder auch ***parse tree***.
- Im ***abstract syntax tree*** (AST) konzentriert man sich auf die relevanten, strukturgebenden Komponenten (hier z.B. Weglassen der Klammern)

Semantik regulärer Ausdrücke

Definition 1.63 (Semantik regulärer Ausdrücke)

Sei Σ ein Alphabet. Die Semantik (Bedeutung) eines regulären Ausdrucks ist eine Abbildung

$$[\![\cdot]\!] : \mathcal{R}_\Sigma \to \mathcal{P}\left(\Sigma^*\right)$$

die jedem regulären Ausdruck eine Sprache zuordnet. Sie ist induktiv definiert durch:

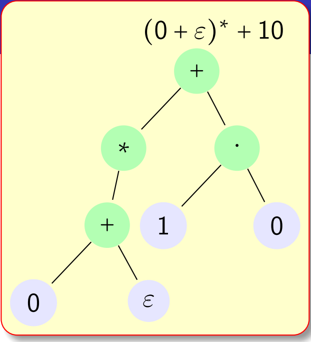
1. $[\![\varnothing]\!] = \varnothing$
2. $[\![\varepsilon]\!] = \{\varepsilon\}$
3. Für alle $a \in \Sigma$: $[\![a]\!] = \{a\}$
4. falls $r_1, r_2 \in \mathcal{R}_\Sigma$, so
   4.1 $[\![(r_1 + r_2)]\!] = [\![r_1]\!] \cup [\![r_2]\!]$
   4.2 $[\![(r_1 \cdot r_2)]\!] = [\![r_1]\!] \cdot [\![r_2]\!]$
5. falls $r \in \mathcal{R}_\Sigma$, so $[\![(r)^*]\!] = [\![r]\!]^*$

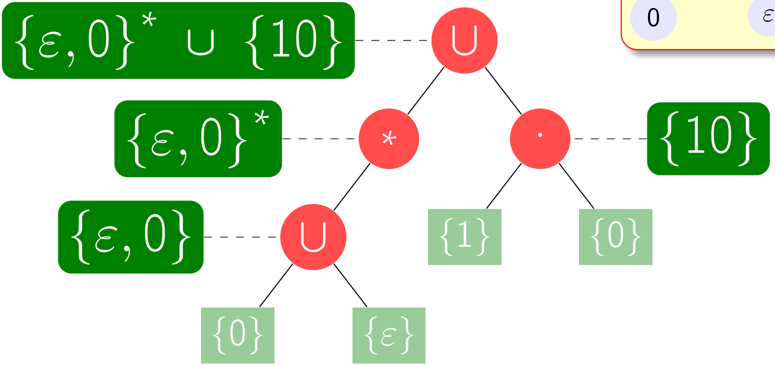
Achtung!

- Sofern eindeutig, schreibt man oft einfach nur r anstelle von $[\![r]\!]$.
- Dann kann man z.B. schreiben: $ab \in \left(a \cdot (a+b)^*\right)$ anstelle von $ab \in [\![\left(a \cdot (a+b)^*\right)]\!]$

usdrücke

1. $[\![\varnothing]\!] = \varnothing$
2. $[\![\varepsilon]\!] = \{\varepsilon\}$
3. Für alle $a \in \Sigma$: $[\![a]\!] = \{a\}$
4. falls $r_1, r_2 \in \mathcal{R}_\Sigma$, so
   4.1 $[\![(r_1 + r_2)]\!] = [\![r_1]\!] \cup [\![r_2]\!]$
   4.2 $[\![(r_1 \cdot r_2)]\!] = [\![r_1]\!] \cdot [\![r_2]\!]$
5. falls $r \in \mathcal{R}_\Sigma$, so $[\![(r)^*]\!] = [\![r]\!]^*$





Beispiel: Semantik regulärer Ausdrücke

Beispiel 1.55 (Semantik von $(0+\varepsilon)^* + 10$)

$$
\begin{aligned}
[\![(0+\varepsilon)^* + 10]\!] &= [\![(0+\varepsilon)^*]\!] \cup [\![10]\!] \\
&= [\![(0+\varepsilon)^*]\!] \cup ([\![1]\!] \cdot [\![0]\!]) \\
&= [\![(0+\varepsilon)^*]\!] \cup (\{1\} \cdot \{0\}) \\
&= [\![(0+\varepsilon)]\!]^* \cup \{10\} \\
&= ([\![0]\!] \cup [\![\varepsilon]\!])^* \cup \{10\} \\
&= (\{0\} \cup \{\varepsilon\})^* \cup \{10\} \\
&= \{0, \varepsilon\}^* \cup \{10\}
\end{aligned}
$$

Reguläre Ausdrücke: Beispiele

Beispiel 1.56 (Reguläre Ausdrücke)

- $0^*10^* = \{w \in \Sigma^*_{Bool} \mid |w|_1 = 1\}$
- $(0+1)^*1(0+1)^* = \{w \in \Sigma^*_{Bool} \mid |w|_1 \geq 1\}$
- $(0+1)^*10010(0+1)^* = \{w \in \Sigma^*_{Bool} \mid w \text{ enthält } 10010 \text{ als Teilwort}\}$
- $((0+1)(0+1)(0+1))^* = \{w \in \Sigma^*_{Bool} \mid |w| \bmod 3 = 0\}$
- $(1+\varepsilon) = \{\varepsilon, 1\}$
- $(0+\varepsilon)1^* = 01^* + 1^*$
- $(0+\varepsilon)(1+\varepsilon) = \{\varepsilon, 0, 1, 01\}$
- $(100+011)^*\varnothing = \varnothing$

Satz von Kleene

Satz 1.1 (Satz von Kleene)

Die Menge der durch reguläre Ausdrücke definierbaren Sprachen entspricht exakt der Menge der regulären Sprachen $\mathcal{L}_{Reg}$.

- **Mit anderen Worten**: Mit endlichen Automaten lassen sich dieselben Sprachen beschreiben, wie mit regulären Ausdrücken.
- Zum Beweis dieser Aussage gibt man Algorithmen an, mit denen man
  - aus einem regulären Ausdruck r einen ε-NEA M konstruieren kann, so dass $L(M) = [\![r]\!]$ bzw.
  - aus einem DEA M einen regulären Ausdruck r gewinnen kann, so dass $L(M) = [\![r]\!]$.

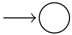
Wir beschränken uns hier auf den ersten Algorithmus

Regulärer Ausdruck in ε-NEA 1

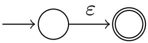
Wie bei der Definition der Semantik, »hangeln« wir uns entlang der induktiven Definition von $\mathcal{R}_\Sigma$:

Induktionsanfang:

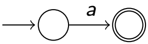
- $r = \varnothing$: ε-NEA M mit $L(M) = [\![\varnothing]\!] = \varnothing$ ist schnell gefunden



- $r = \varepsilon$: ε-NEA M mit $L(M) = [\![\varepsilon]\!] = \{\varepsilon\}$ ist ebenfalls schnell gefunden:



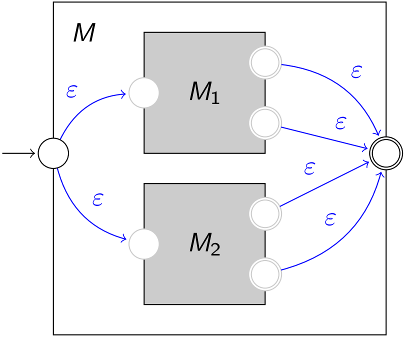
- $r = a$: für $a \in \Sigma$: ε-NEA mit $L(M) = \{a\}$ ist auch leicht gefunden:



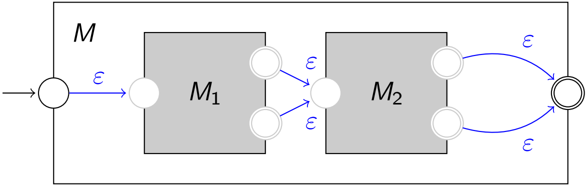
Für zusammengesetzte Ausdrücke greifen wir auf die Konstruktionen zurück, die wir bei der Diskussion der Abschlusseigenschaften von regulären Sprachen vorgestellt haben:

Induktionsschritt:

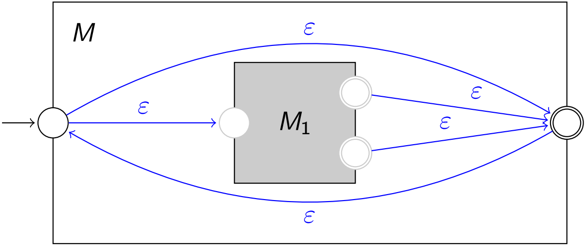
- $r = (r_1 + r_2)$: konstruiere M_1 mit $L(M_1) = [\![r_1]\!]$ und M_2 mit $L(M_2) = [\![r_2]\!]$ - verbinde diese wie folgt zu einem ε-NEA M mit $L(M) = [\![r_1]\!] \cup [\![r_2]\!]$:



- $r = (r_1 \cdot r_2)$: konstruiere M_1 mit $L(M_1) = [\![r_1]\!]$ und M_2 mit $L(M_2) = [\![r_2]\!]$ - verbinde diese wie folgt zu einem ε-NEA M mit $L(M) = [\![r_1]\!] \cdot [\![r_2]\!]$:

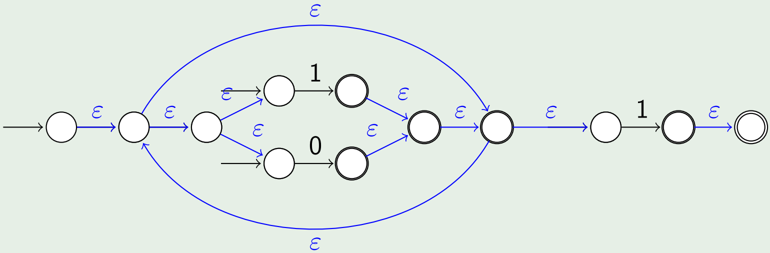


- $r = (r_1)^*$: konstruiere M_1 mit $L(M_1) = [\![r_1]\!]$ - konstruiere daraus einem ε-NEA M mit $L(M) = [\![r_1]\!]^*$:



Beispiel: Regulärer Ausdruck in ε-NEA

Beispiel 1.57 ($r = (0+1)^* \cdot 1$)



Offenbar sind viele Vereinfachungen möglich.

Stats

- definition: 1.63
- example: 1.57
- solution: 1.5