Reguläre Sprachen, Ausdrucksstärke (Teil 1)

BC George (HSBI)

Motivation

Was muss ein Compiler wohl als erstes tun?

- Einlesen
- Bestandteile
  - Wörter
  - Zeichen

Themen für heute

- Lexer Scanner Zerteiler
- Endliche Automaten
- Reguläre Sprachen

Endliche Automaten

Alphabete

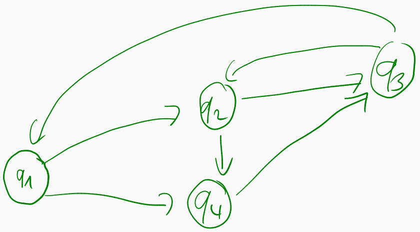
Def.: Ein **Alphabet** Σ ist eine endliche, nicht-leere Menge von Symbolen. Die Symbole eines Alphabets heißen *Buchstaben*.

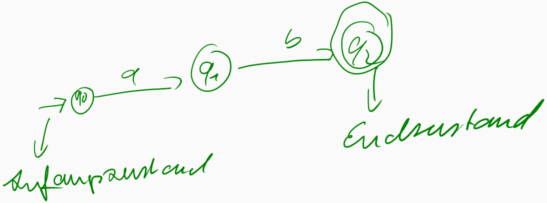
epsilon

Def.: Ein **Wort** w *über einem Alphabet* Σ ist eine endliche Folge von Symbolen aus Σ. ϵ ist das leere Wort. Die *Länge* $|w|$ eines Wortes w ist die Anzahl von Buchstaben, die es enthält (Kardinalität).

Def.: Eine **Sprache** L *über einem Alphabet* Σ ist eine Menge von Wörtern über diesem Alphabet. Sprachen können endlich oder unendlich viele Wörter enthalten.

State machine





Bestimmte State machines:

- Eingaben bestimmen Zustandsübergänge
- Zustandsübergänge sind eindeutig
- Es gibt Anfang(szustand) und End(zuständ)e

genau einen

Automat läuft einmal durch

$Q = \{q_0, q_1, q_2\}$

Wie definieren wir das formal?

- Menge der Zustände
- Eingabealphabet
- Anfangszustand
- Endzustände
- Übergangsfkt

großes Sigma

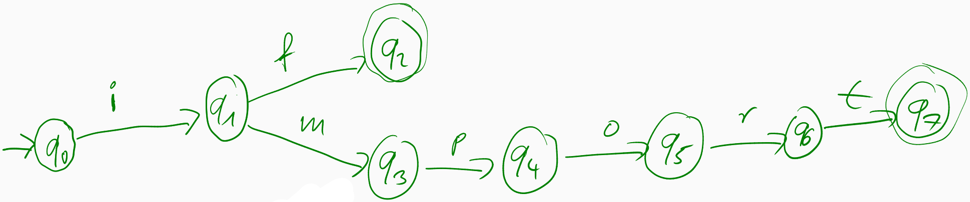
Def.: Ein **deterministischer endlicher Automat** (DFA) ist ein 5-Tupel $A = (Q, \Sigma, \delta, q_0, F)$ mit

delta

- Q : endliche Menge von **Zuständen**
- Σ : Alphabet von **Eingabesymbolen**
- δ : die (eventuell partielle) **Übergangsfunktion** $(Q \times \Sigma) \to Q$, δ kann partiell sein
- $q_0 \in Q$: der **Startzustand**
- $F \subseteq Q$: die Menge der **Endzustände**

$\mathbb{N} \to \mathbb{R}$

$x \mapsto x+1$



ifm → nicht akzeptiert

impo → "

$$\delta^*(q_0, if) = \delta(\underbrace{\delta^*(q_0, i)}_{\delta(q_0, i)}, f)$$

q_0

$\delta(q_0, i)$

0 mal oder endlich oft angewendet

Def.: Wir definieren $\delta^* : (Q \times \Sigma^*) \to Q$: induktiv wie folgt:

$a_i \in \Sigma \quad w = a_1 \dots a_n$

- Basis: $\delta^*(q, \epsilon) = q \ \forall q \in Q$
- Induktion: $\delta^*(q, a_1, \dots, a_n) = \delta(\underbrace{\delta^*(q, a_1, \dots, a_{n-1})}_{\text{Zustand nach der Verarbeitung von } a_1 \dots a_{n-1}}, a_n)$

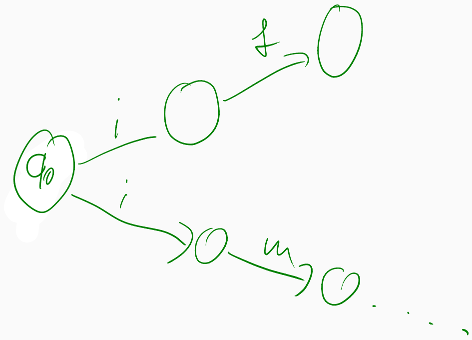
Def.: Ein DFA akzeptiert ein Wort $w \in \Sigma^*$ genau dann, wenn $\delta^*(q_0, w) \in F$.

$q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_5 \xrightarrow{a_3} q_7$

Beispiel

s. Folie 9

Nichtdeterministische endliche Automaten



Def.: Nichtdeterministischer Automat

Def.: Ein **nichtdeterministischer endlicher Automat** (NFA) ist ein 5-Tupel $A = (Q, \Sigma, \delta, q_0, F)$ mit

- Q : endliche Menge von **Zuständen**
- Σ : Alphabet von **Eingabesymbolen**
- δ : die (eventuell partielle) **Übergangsfunktion** $(Q \times \Sigma) \to \cancel{Q}$ $\mathcal{P}(Q)$ Menge aller Teilmengen von Q
- $q_0 \in Q$: der **Startzustand**
- $F \subseteq Q$: die Menge der **Endzustände**

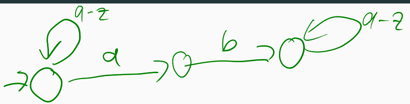
Akzeptierte Sprachen

Def.: Sei A ein DFA oder ein NFA. Dann ist **L(A)** die von A akzeptierte Sprache, d. h.

$L(A) = \{\text{Wörter } w \mid \delta^*(q_0, w) \in F\}$

$w = a_1 \ldots a_n \quad a_i \in \Sigma$

Wozu NFAs im Compilerbau?



Pattern: ab

Pattern Matching (Erkennung von Schlüsselwörtern, Bezeichnern, . . .) geht mit NFAs.

NFAs sind so nicht zu programmieren, aber:

Satz: Eine Sprache L wird von einem NFA akzeptiert $\Leftrightarrow$ L wird von einem DFA akzeptiert.

D. h. es existieren Algorithmen zur

- Umwandlung von NFAs in DFA~~S~~s
- Minimierung von DFAs

Reguläre Sprachen

Reguläre Ausdrücke definieren Sprachen

Def.: Induktive Definition von **regulären Ausdrücken** (regex) und der von ihnen repräsentierten Sprache **L**:

- Basis:
  - ϵ und $\emptyset$ sind reguläre Ausdrücke mit $L(\epsilon) = \{\epsilon\}$, $L(\emptyset) = \emptyset$ — leere Sprache
  - Sei a ein Symbol $\Rightarrow$ a ist ein regex mit $L(a) = \{a\}$
- Induktion: Seien $E,\ F$ reguläre Ausdrücke. Dann gilt:
  - $E + F$ ist ein regex und bezeichnet die Vereinigung $L(E + F) = L(E) \cup L(F)$ $= |$ oder,
  - EF ist ein regex und bezeichnet die Konkatenation $L(EF) = L(E)L(F)$
  - E^* ist ein regex und bezeichnet die Kleene-Hülle $L(E^*) = (L(E))^*$ 0 oder unendlich oft
  - (E) ist ein regex mit $L((E)) = L(E)$ $(ab)^* = \{\varepsilon, ab, abab, ababab, \ldots\}$

Vorrangregeln der Operatoren für reguläre Ausdrücke: *, Konkatenation, +

$(E^+$: endlich oft$)$

Beispiel

$1(0-9)^*2$ beschreibt alle Wörter, die mit 1 beginnen und mit 2 enden, dazwischen Ziffern 0-9, 0 oder endlich viele

0^* ε oder endlich viele Nullen

Wichtige Identitäten

Automaten akzeptieren Sprachen,
regex beschreiben Sprachen

$r_1 = if$
$r_2 = while$
$r_3 = import$
$\vdots$
$r_i = \{$

Satz: Sei A ein DFA $\Rightarrow \exists$ regex R mit $L(A) = L(R)$.

Satz: Sei E ein regex $\Rightarrow \exists$ DFA A mit $L(E) = L(A)$.

if a < 10   if a 10 <

Trennzeichen: Delimiter

Syntax einer Sprache

gehört über nach

Start-symbol

Satz → S P O

S → Artikel Substantiv

Artikel → 'The' | 'A' | 'An'

Substantiv → 'street' | 'book' | 'house' | 'door' . . .

oder oder

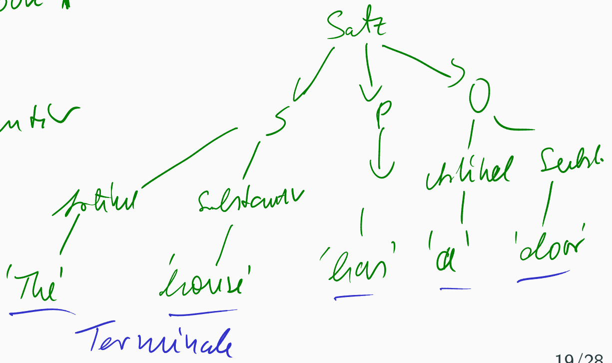
P → 'has' | 'is'

O → Artikel Substantiv

Nichtterminale

Produktionen

Ableitungsbaum



Formale Definition formaler Grammatiken

Def.: Eine *formale Grammatik* ist ein 4-Tupel $G = (N, T, P, S)$ aus

- N: endliche Menge von **Nichtterminalen**
- T: endliche Menge von **Terminalen**, $N \cap T = \emptyset$
- $S \in N$: **Startsymbol**
- P: endliche Menge von **Produktionen** der Form

  $X \rightarrow Y$ mit $X \in (N \cup T)^* N (N \cup T)^*, Y \in (N \cup T)^*$

Ableitungen

Def.: Sei $G = (N, T, P, S)$ eine Grammatik, sei $\alpha A \beta$ eine Zeichenkette über $(N \cup T)^*$ und sei $A \to \gamma$ eine Produktion von G.

Wir schreiben: $\alpha A \beta \Rightarrow \alpha \gamma \beta$ ($\alpha A \beta$ leitet $\alpha \gamma \beta$ ab).

eine Produktion anwenden

Nichtterminale: Großbuchstaben

Def.: Wir definieren die Relation $\overset{*}{\Rightarrow}$ induktiv wie folgt:

- Basis: $\forall \alpha \in (N \cup T)^* \alpha \overset{*}{\Rightarrow} \alpha$ (Jede Zeichenkette leitet sich selbst ab.)
- Induktion: Wenn $\alpha \overset{*}{\Rightarrow} \beta$ und $\beta \Rightarrow \gamma$ dann $\alpha \overset{*}{\Rightarrow} \gamma$

Def.: Sei $G = (N, T, P, S)$ eine formale Grammatik. Dann ist $L(G) = \{$Wörter w über $T \mid S \overset{*}{\Rightarrow} w\}$ die von G erzeugte Sprache.

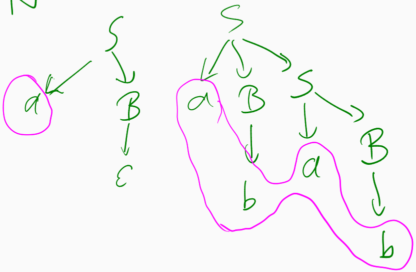
Grammatiken erzeugen Sprachen

Beispiel

oder

$$P \begin{cases} S \rightarrow aBS \mid aB \\ B \rightarrow aB \mid b \mid \varepsilon \end{cases}$$

$$G = (\underbrace{\{S, B\}}_{N}, \underbrace{\{a, b\}}_{T}, P, S)$$



Reguläre Grammatiken

Def.: Eine **reguläre (oder type-3-) Grammatik** ist eine formale Grammatik mit den folgenden Einschränkungen:

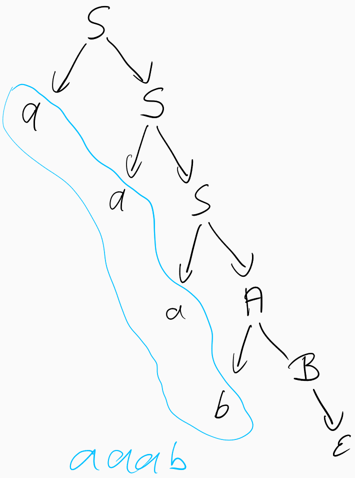
- Alle Produktionen sind entweder von der Form
  - $X \to aY$ mit $X \in N, a \in T, Y \in N$ (*rechtsreguläre* Grammatik) oder exor
  - $X \to Ya$ mit $X \in N, a \in T, Y \in N$ (*linksreguläre* Grammatik)
- $X \to \epsilon$ ist erlaubt

Beispiel

$S \to aS \mid aA$

$A \to bB$

$B \to \varepsilon$



Reguläre Sprachen

Satz: Die von endlichen Automaten akzeptiert Sprachklasse, die von regulären Ausdrücken beschriebene Sprachklasse und die von regulären Grammatiken erzeugte Sprachklasse sind identisch und heißen **reguläre Sprachen**.

Wrap-Up

Wrap-Up

- Definition und Aufgaben von Lexern
- DFAs und NFAs
- Reguläre Ausdrücke
- Reguläre Grammatiken
- Zusammenhänge zwischen diesen Mechanismen und Lexern, bzw. Lexergeneratoren

Last modified: *64341f7 (lecture: split regular expressions into two lessons, 2025-08-29)*