

# Reguläre Sprachen, Ausdrucksstärke

---

BC George (HSBI)

Unless otherwise noted, this work is licensed under CC BY-SA 4.0.

# Motivation

---

Was muss ein Compiler wohl als erstes tun?

# Themen für heute

- Endliche Automaten
- Reguläre Sprachen

# Endliche Automaten

---

# Deterministische endliche Automaten

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

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

# Nichtdeterministische endliche Automaten

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

- $Q$  : endliche Menge von **Zuständen**
- $\Sigma$  : Alphabet von **Eingabesymbolen**
- $\delta$  : die (eventuell partielle) Übergangsfunktion  $(Q \times \Sigma) \rightarrow Q$
- $q_0 \in Q$  : der **Startzustand**
- $F \subseteq Q$  : die Menge der **Endzustände**

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

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



# Wozu NFAs im Compilerbau?

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 DFAS
- Minimierung von DFAs

# Reguläre Sprachen

---

# Reguläre Ausdrücke

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

- Basis:
  - $\epsilon$  und  $\emptyset$  sind reguläre Ausdrücke mit  $L(\epsilon) = \{\epsilon\}$ ,  $L(\emptyset) = \emptyset$
  - 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  $L(E + F) = L(E) \cup L(F)$
  - $EF$  ist ein regex und bezeichnet die Konkatenation  $L(EF) = L(E)L(F)$
  - $E^*$  ist ein regex und bezeichnet die Kleene-Hülle  $L(E^*) = (L(E))^*$
  - $(E)$  ist ein regex mit  $L((E)) = L(E)$

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

**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  $A \rightarrow \gamma$  eine Produktion von  $G$ .

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

**Def.:** Wir definieren die Relation  $\Rightarrow^*$  induktiv wie folgt:

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

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

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

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

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

## Reguläre Sprachen

- einfache Struktur
- Matchen von Symbolen (z. B. Klammern) nicht möglich, da die fixe Anzahl von Zuständen eines DFAs die Erkennung solcher Sprachen verhindert.

# Wozu reguläre Sprachen im Compilerbau?

- Reguläre Ausdrücke
  - definieren Schlüsselwörter und alle weiteren Symbole einer Programmiersprache, z. B. den Aufbau von Gleitkommazahlen
  - werden (oft von einem Generator) in DFAs umgewandelt
  - sind die Basis des *Scanners* oder *Lexers*



# Ein Lexer ist mehr als ein DFA

- Ein **Lexer**

- wandelt mittels DFAs aus regulären Ausdrücken die Folge von Zeichen der Quelldatei in eine Folge von sog. Token um
- bekommt als Input eine Liste von Paaren aus regulären Ausdrücken und Tokennamen, z. B. ("while", WHILE)
- Kommentare und Strings müssen richtig erkannt werden. (Schachtelungen)
- liefert Paare von Token und deren Werte, sofern benötigt, z. B. (WHILE, \_), oder (IDENTIFIER, "radius") oder (INTEGERZAHL, "334")

# Wie geht es weiter?

- Ein **Parser**

- führt mit Hilfe des Tokenstreams vom Lexer die Syntaxanalyse durch
- basiert auf einer sog. kontextfreien Grammatik, deren Terminale die Token sind
- liefert die syntaktische Struktur in Form eines Ableitungsbaums (**syntax tree**, **parse tree**), bzw. einen **AST** (abstract syntax tree) ohne redundante Informationen im Ableitungsbaum (z. B. Semikolons)
- liefert evtl. Fehlermeldungen

## Wrap-Up

---

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

# LICENSE



Unless otherwise noted, this work is licensed under CC BY-SA 4.0.