Reguläre Sprachen, Ausdrucksstärke

BC George (FH Bielefeld)

Motivation

Was muss ein Compiler wohl als erstes tun?

Themen für heute

- Endliche Automaten
- Reguläre Ausdrücke

Endliche Automaten

Alphabete

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

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.: $\Sigma^k = \{w \text{ über } \Sigma \mid |w| = k\}$

$\Sigma^* = \bigcup_{i \in \mathbb{N}_0} \Sigma^i$ (die Kleene-Hülle von Σ)

$\Sigma^+ = \bigcup_{i \in \mathbb{N}} \Sigma^i$

Sprachen über Alphabete

Def.: Seien $x = a_1 a_2 \ \ldots \ a_n$ und $y = b_1 b_2 \ \ldots \ b_m$ Wörter. Wir nennen $xy = x \circ y = a_1 \ \ldots \ a_n b_1 \ \ldots \ b_m$ die *Konkatenation* von x und y.

Def.: *Eine Sprache L über einem Alphabet Σ* ist eine Teilmenge von $\Sigma^* : \ L \subseteq \Sigma^*$

Deterministische endliche Automaten

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

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

Die Übergangsfunktion

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

- Basis: $\delta^*(q, \epsilon) = q \ \forall q \in Q$
- Induktion: $\delta^*(q, a_1, \ldots, a_n) = \delta(\delta^*(q, a_1, \ldots, a_{n-1}), a_n)$

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

Def.: Die Sprache eines DFA A $L(A)$ ist definiert durch:

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

Beispiel

Nichtdeterministische endliche Automaten

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

- Q: eine endliche Menge von Zuständen
- Σ: ein Alphabet von Eingabesymbolen
- δ: die Übergangsfunktion $(Q \times \Sigma) \to \mathcal{P}(Q)$
- $q_0 \in Q$: der Startzustand
- $F \subseteq Q$: die Menge der Endzustände

Die Übergangsfunktion eines NFAs

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

- Basis: $\delta^*(q, \epsilon) = q \ \forall q \in Q$
- Induktion: Sei $w \in \Sigma^*, w = xa, x \in \Sigma^*, a \in \Sigma$ mit

  $\delta^*(q, x) = \{p_1, \ \ldots, \ p_k\}, p_i \in Q$, sei

  $A = \bigcup_{i=1}^{k} \delta(p_i, a) = \{r_1, \ldots r_m\}, r_j \in Q.$

  Dann ist $\delta^*(q, w) = \{r_1, \ \ldots \ , r_m\}$.

Wozu NFAs im Compilerbau?

Pattern Matching 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.

Konvertierung eines NFAs in einen DFA

Gegeben: Ein NFA $A = (Q, \Sigma, \delta, q_0, F)$

Wir konstruieren einen DFA $A' = (Q', \Sigma, \delta', q_0, F')$ wie folgt:

```
Q' = {{q0}}
for each q in Q':
  for each a ∈ Σ:
    n = {p | δ(r, a) = p, r ∈ q}
    Q' = Q' ∪ {n}
    δ'(q,a) = n
F' = {q ∈ Q' | q ∩ F ≠ ∅}
{p,q} ->  pq //Umbenennung
```

Abbildung 1: Konvertierung NFA in DFA

Beispiel

δ	a	b
$\rightarrow q_0$	$\{q_0\}$	$\{q_1, q_2\}$
q_1	$\{q_2\}$	$\{q_1\}$
$*q_2$	-	$\{q_0, q_2\}$

δ'	a	b
$\rightarrow \{q_0\}$	$\{q_0\}$	$\{q_1, q_2\}$
$*\{q_1 q_2\}$	$\{q_2\}$	$\{q_0, q_1, q_2\}$
$*\{q_2\}$	-	$\{q_0, q_2\}$
$*\{q_0, q_2\}$	$\{q_0\}$	$\{q_0, q_1, q_2\}$
$*\{q_0, q_1, q_2\}$	$\{q_0, q_2\}$	$\{q_0, q_1, q_2\}$

Minimierung eines DFAs

Ist ist der DFA A nicht vollständig, wird ein Fehlerzustand q_e, der kein Endzustand ist, hinzugefügt und in alle leeren Tabellenfelder eingetragen.

Dann wird eine Matrix generiert, die für alle Zustandspaare sagt, ob die beiden Zustände zu einem verschmelzen können.

```
for each (p,q) with p,q ∈ Q × Q, p ≠ q:
  if (p ∈ F and q ∉ F) or (p ∉ F and q ∈ F):
    D(p,q) = "-"
  else
    D(p,q) = ϵ

repeat
  for each (q,p) ∈ Q × Q with p ≠ q:
    for each a ∈ Σ:
      if D(p,q) = ϵ and
          D(δ(p,a), δ(q,a)) ≠ ϵ:
        D(p,q) = a
until there are no changes

for each entry in D with D(p,q) = ϵ:
  combine p and q to one state
```

Abbildung 2: DFA Minimierung

Reguläre Ausdrücke

Operatoren auf Sprachen

Def.: Seien L und M Sprachen.

- $L \cup M = \{w \mid w \in L \vee w \in M\}$
- $LM = L \cdot M = L \circ M = \{vw \mid v \in L \wedge w \in M\}$
- Die Kleene-Hülle einer Sprache:
  - Basis: $L^0 = \{\epsilon\}$
  - Induktion: $L^i = \{xw \mid x \in L^{i-1}, w \in L, i > 0\}$,
    $L^* = \bigcup_{i \geq 0} L^i$,
    $L^+ = \bigcup_{i > 0} L^i$

Reguläre Ausdrücke

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

- Basis:
  - ϵ 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, $+$

Wichtige Identitäten

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

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

Beispiel: Umwandlung eines regex in einen NFA

Formale Grammatiken

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

- N: einer endlichen Menge von *Nichtterminalen*
- T: einer endlichen Menge von *Terminalen*, $N \cap T = \emptyset$
- $S \in N$: dem *Startsymbol*
- P: einer endlichen Menge von *Produktionen* der Form: $X \to Y$ 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 sagen: $\alpha A \beta \Rightarrow \alpha \gamma \beta$ ($\alpha A \beta$ leitet $\alpha \gamma \beta$ ab).

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 $L(G) = \{w \in T^* \mid S \overset{*}{\Rightarrow} w\}$ die von G erzeugte Sprache.

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
  - $X \to Ya$ mit $X \in N, a \in T, Y \in N$ (*linksreguläre* Grammatik)
- $X \to \epsilon$ ist in beiden Fällen erlaubt.

Reguläre Sprachen

Satz: Die von rechtsregulären Grammatiken erzeugten Sprachen sind genau die von linksregulären Grammatiken erzeugten Sprachen. Beide werden *reguläre* Sprachen genannt.

Satz: Die von regulären Ausdrücken beschriebenen Sprachen sind die regulären Sprachen.

Das Pumping Lemma für reguläre Sprachen

Satz: Das *Pumping Lemma für reguläre Sprachen*:

Sei L eine reguläre Sprache.

$\Rightarrow \exists$ Konstante $n \in \mathbb{N}$:

$$\forall_{\substack{w \in L \\ |w| \geq n}} \exists x, y, z \in \Sigma^* \text{ mit } w = xyz, y \neq \epsilon, |xy| \leq n :$$

$$\forall_{k \geq 0} xy^k z \in L$$

Abschlusseigenschaften regulärer Sprachen

Die Klasse der regulären Sprachen ist abgeschlossen unter

- Vereinigung
- Konkatenation
- Kleene-Stern
- Komplementbildung
- Durchschnitt

Entscheidbarkeit für reguläre Sprachen

Satz: Es ist entscheidbar,

- ob eine gegebene reguläre Sprache leer ist
- ob $w \in \Sigma^*$ in einer gegebenen regulären Sprache enthalten ist (Das "Wort-Problem")
- ob zwei reguläre Sprachen äquivalent sind

Grenzen der regulären Sprachen

Reguläre Sprachen sind von ihrer Struktur her einfach. Schon Sprachen, in denen etwas “gematcht” werden muss, lassen sich nicht mehr regulär beschreiben, weil z. B. die fixe Anzahl von Zuständen eines DFAs die Erkennung solcher Sprachen verhindert.

Wozu das Ganze?

Im Compilerbau werden reguläre Ausdrücke benutzt, um die Schlüsselwörter und weitere Symbole der zu erkennenden Sprache anzugeben. Daraus wird mit Hilfe eines Generators, der aus den regulären Ausdrücken DFAs (oder einen großen DFA) macht, der sog. Scanner oder Lexer genannt, generiert. Seine Aufgabe ist es, die Folge von Zeichen in der Quelldatei in eine Folge von sog. Token umzuwandeln. Z. B. wird so aus den Zeichen des Schlüsselwortes *while* im Programmtext das Token für *while* gemacht, das in der Syntaxanalyse weiterverarbeitet wird. Die Tokenfolge eines Programms ist ein Wort einer Sprache, die der Parser erkennt. Jedes vom Lexer erkannte Token ist dort also ein terminales Symbol.

Ein Lexer ist mehr als ein DFA

Was ist zu beachten:

- Man braucht mindestens eine Liste von Paaren aus regulären Ausdrücken und Tokennamen.
- Neben den Schlüsselwörtern und Symbolen wie (,), *, ... müssen auch Namen für Variablen, Funktionen, Klassen, Methoden, ... (sog. Identifier) erkannt werden
- Namen haben meist eine gewisse Struktur, die sich mit regulären Ausdrücken beschreiben lassen.
- Erlaubte Token sind in der Grammatik des Parsers beschrieben, d. h. für literale Namen, Strings, Zahlen liefert der Scanner zwei Werte:
  - z. B. <ID, “radius”>, <Integerzahl, 558>
- Kommentare und Strings müssen richtig erkannt werden. (Schachtelungen)

Man kann natürlich auch einen Lexer selbst programmieren, d. h. die DFAs für die regulären Ausdrücke implementieren.

Automatisch oder händisch

Wrap-Up

Wrap-Up

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

LICENSE