universität innsbruck

Einführung in die Theoretische Informatik

Martin Avanzini Christian Dalvit Jamie Hochrainer
Georg Moser Johannes Niederhauser Jonas Schöpf

`https://tcs-informatik.uibk.ac.at`

universität innsbruck

Zusammenfassung

Definition (Deterministischer endlicher Automat (kurz: DEA))

Ein DEA ist ein 5-Tupel $A = (Q, \Sigma, \delta, q_0, F)$ sodass

1. Q eine endliche Menge von Zuständen
2. Σ eine endliche Menge von Eingabesymbole
3. $\delta \colon Q \times \Sigma \to Q$ die Übergangsfunktion
4. $q_0 \in Q$ der Startzustand
5. $F \subseteq Q$ eine endliche Menge von akzeptierenden Zuständen

Zu beachten: δ muss für alle möglichen Argumente definiert sein

Satz

Sei A ein DEA, dann ist $\mathrm{L}(A)$ regulär und umgekehrt existiert zu jeder regulären Sprache L ein DEA A, sodass $L = \mathrm{L}(A)$

Einführung in die Logik

Syntax & Semantik der Aussagenlogik, Formales Beweisen, Konjunktive und Disjunktive Normalformen

Einführung in die Algebra

Algebraische Strukturen, Boolesche Algebra, Universelle Algebra

Einführung in die Theorie der Formalen Sprachen

Grammatiken und Formale Sprachen, Chomsky-Hierarchie, Reguläre Sprachen, Kontextfreie Sprachen, Anwendungen von formalen Sprachen

Einführung in die Berechenbarkeitstheorie und Komplexitätstheorie

Algorithmisch unlösbare Probleme, Turing Maschinen, Registermaschinen, Komplexitätstheorie

Einführung in die Programmverifikation

Prinzipien der Analyse von Programmen, Verifikation nach Hoare

universität innsbruck

Linksableitung, Rechtsableitung und Rekursive Inferenz

Definition

- Eine Linksableitung ist eine Ableitung sodass immer die am weitesten links stehende Variable ersetzt wird $\underset{\ell}{\Rightarrow}, \overset{*}{\underset{\ell}{\Rightarrow}}$
- In einer Rechtsableitung wird immer die am weitesten rechts stehende Variable ersetzt $\underset{r}{\Rightarrow}, \overset{*}{\underset{r}{\Rightarrow}}$

Beispiel

Wir betrachten KFG $G = (\{S\}, \{(,)\}, R, S)$, wobei R:

$$S \to \epsilon \mid (S) \mid SS$$

dann gilt $S \underset{r}{\Rightarrow} SS \underset{r}{\Rightarrow} S(S) \underset{r}{\Rightarrow} S() \underset{r}{\Rightarrow} (S)() \underset{r}{\Rightarrow} ()()$

Definition (Eindeutigkeit einer Grammatik)

KFG G heißt eindeutig, wenn jedes Wort $x \in L(G)$ genau eine Linksableitung besitzt, ansonsten mehrdeutig

Definition

Sei G eine KFG und sei $A \to x$ eine Regel in G, sodass $x \in (V \cup \Sigma)^*$. Die Sprache von A, bezeichnet als $\mathrm{L}(A)$, kann mittels rekursiver Inferenz definiert werden:

1. Wenn $x \in \Sigma^*$, dann $x \in \mathrm{L}(A)$
2. Andererseits, sei $x = X_1 \cdots X_n$, wobei $X_i \in V \cup \Sigma$

   Gelte außerdem $x_i \in \mathrm{L}(X_i)$ oder $x_i = X_i \in \Sigma$

   Dann gilt $x_1 x_2 \cdots x_n \in \mathrm{L}(A)$

Bemerkung

- Die rekursive Inferenz entsprich dem bottom-up parsing eines Compilers: zuerst wird der Rumpf einer Regel gelesen, dann wird das Ergebnis der Variable im Kopf übergeben
- Der Parsergenerator `yacc` (siehe später) generiert einen bottom-up parser

Beispiel

Wir betrachten die KFG $G_1 = (\{E, I\}, \{+, \cdot, (,), \mathrm{a}, \mathrm{b}, 0, 1\}, R, E)$, wobei R wie folgt:

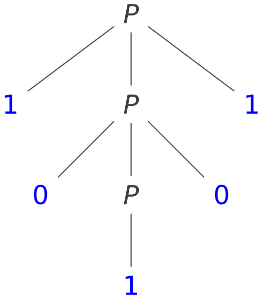
$$
\begin{array}{llll}
E \to I & E \to E \cdot E & I \to \mathrm{a} & I \to I0 \\
E \to E+E & E \to (E) & I \to I\mathrm{b} & I \to I1 \\
 & I \to I\mathrm{a} & I \to \mathrm{b} &
\end{array}
$$

und zeigen $(\mathrm{a}+\mathrm{b}10) \in \mathrm{L}(E)$:

	Wort x	Variable	Regel	Rekursion
1	a	I	$I \to \mathrm{a}$	
2	b	I	$I \to \mathrm{b}$	
3	b1	I	$I \to I1$	2
4	b10	I	$I \to I0$	3
5	a	E	$E \to I$	1
6	b10	E	$E \to I$	4
7	a+b10	E	$E \to E+E$	5, 6
8	(a+b10)	E	$E \to (E)$	7

Wiederholung: Bäume

Das ist ein Baum



Und so sehen Bäume jetzt aus

Definition (Syntaxbaum)

Ein Syntaxbaum für KFG $G = (V, \Sigma, R, S)$ ist ein Baum B mit:

1. Jeder innere Knoten von B ist eine Variable in V
2. Jedes Blatt in B ist entweder:
   - ein Terminal aus Σ
   - ein Nichtterminal aus V, oder
   - ϵ

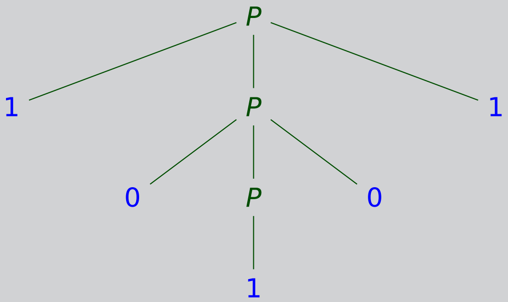
   Im letzten Fall ist das Blatt das einzige Kind seines Vorgängers
3. Sei A ein innerer Knoten, $X_1, \ldots, X_n$ seine Kinder ($X_i \in V \cup \Sigma$); dann gilt:

$$A \to X_1 \cdots X_n \in R$$

Das Ergebnis eines Syntaxbaums B für G ist das Wort über $(V \cup \Sigma)^*$, das wir erhalten, wenn wir die Blätter in B von links nach rechts lesen

Beispiel

Wir betrachten die KFG $G = (\{P\}, \Sigma, R, P)$ mit den Regeln R: $P \to \epsilon \mid 0 \mid 1 \mid 0P0 \mid 1P1$
Dann ist der folgende Baum ein Syntaxbaum für G:

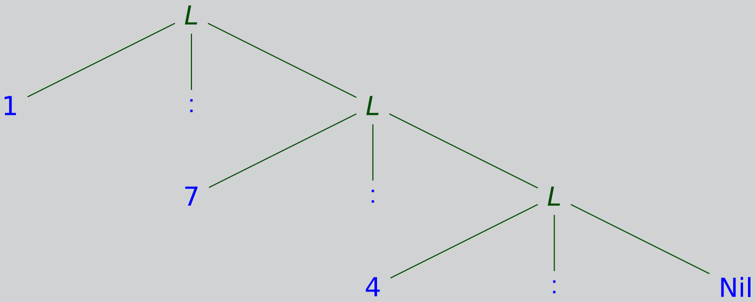


Bemerkung

Vergleiche AST zum Parsen von Haskellausdrücken in Funktionaler Programmierung

Beispiel

Sei $G = (\{L\}, \{0, 1, \ldots, 9, \mathrm{Nil}, :\}\, R, P)$ mit den Regeln R: $L \to \mathrm{Nil} \mid 0 : L \mid 1 : L \mid \cdots \mid 9 : L$



Das entspricht der Liste $1 : 7 : 4 : \mathrm{Nil}$

Bemerkung

Eigenschaften von Syntaxbäumen können mit Induktion über die Höhe des Baumes gezeigt werden.

Satz

Sei Σ ein Alphabet und sei $x \in \Sigma^*$. Die folgenden Beschreibungen kontextfreier Sprachen sind äquivalent:

1. $x \in \mathrm{L}(A)$ *nach dem rekursiven Inferenzverfahren*
2. $A \overset{*}{\Rightarrow} x$
3. $A \underset{\ell}{\overset{*}{\Rightarrow}} x$
4. $A \underset{r}{\overset{*}{\Rightarrow}} x$
5. *Es existiert ein Syntaxbaum mit Wurzel A und Ergebnis x*

$$\begin{tikzcd}
\boxed{x \in \mathrm{L}(A)} \arrow[r, Rightarrow] & \boxed{\exists \text{ Syntaxbaum}} \arrow[d, Rightarrow] \\
\boxed{A \overset{*}{\Rightarrow} x} \arrow[u, Rightarrow] & \boxed{A \underset{\ell}{\overset{*}{\Rightarrow}} x} \arrow[l, Rightarrow]
\end{tikzcd}$$

Satz

Angenommen $x \in \mathrm{L}(A)$ nach dem rekursiven Inferenzverfahren, dann gibt es einen Syntaxbaum B mit Wurzel A und Ergebnis x

Beweis.

Mit Induktion nach der Anzahl der Rekursionen im Inferenzverfahren

- Basis $x \in \mathrm{L}(A)$, $x = a_1 \ldots a_k$ benutzt genau die Regel

  $$A \to a_1 \ldots a_k$$

  Wir konstruieren einen Syntaxbaum B mit Wurzel A wie folgt

  $$A$$

  $$a_1 \ldots a_k$$

Beweis (Fortsetzung).

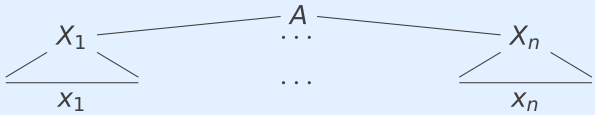
- Schritt $x \in \mathrm{L}(A)$, $x = x_1 \ldots x_n$ benutzt die Regel

$$A \to X_1 \cdots X_n$$

wobei $X_i \in V \cup \Sigma$ und $\forall\, i\; x_i \in \mathrm{L}(X_i)$

Nach IH $\exists$ Syntaxbäume B_i gelten mit Wurzeln X_i und Ergebnis x_i

Wir konstruieren einen neuen Syntaxbaum mit Wurzel A:



∎

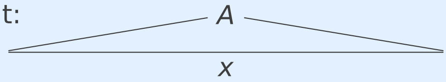
Satz

Sei B ein Syntaxbaum mit Wurzel A und Ergebnis $x \in \Sigma^*$, dann gibt es eine Linksableitung (eine Rechtsableitung) von x aus A

Beweis.

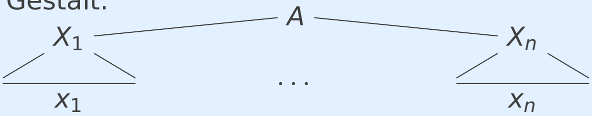
Mit Induktion nach der Höhe des Syntaxbaums B

- Basis Habe B die Gestalt:



  dann existiert $A \to x \in R$, also $A \underset{\ell}{\Rightarrow} x$

- Schritt Habe B die Gestalt:



  dann $\exists\ A \to X_1 \cdots X_n \in R$, also:

$$A \underset{\ell}{\Rightarrow} X_1 \cdots X_n \overset{*}{\underset{\ell}{\Rightarrow}} x_1 X_2 \cdots X_n \overset{*}{\underset{\ell}{\Rightarrow}} x_1 \cdots x_{n-1} x_n$$

∎

Satz

Angenommen $A \stackrel{*}{\Rightarrow} x$ mit $x \in \Sigma^*$, dann liefert das rekursive Inferenzverfahren, dass $x \in \mathrm{L}(A)$

Beweis.

Mit Induktion nach der Länge ℓ der Ableitung $A \stackrel{*}{\Rightarrow} x$:

- Basis Sei $\ell = 1$, dann gilt $x \in \mathrm{L}(A)$
- Schritt Angenommen $\ell = \ell' + 1$
  Zunächst können wir die Ableitung $A \stackrel{*}{\Rightarrow} x$ wie folgt schreiben:

$$A \Rightarrow X_1 X_2 \cdots X_n \stackrel{*}{\Rightarrow} x_1 x_2 \cdots x_n = x$$

  Wir verwenden, dass wir die Ableitungen der Sätze x_i aufbrechen können, also gilt:
  - Wenn $X_i \in \Sigma$, dann $X_i = x_i$
  - Wenn $X_i \in V$, dann gilt $X_i \stackrel{*}{\Rightarrow} x_i$ und somit $x_i \in \mathrm{L}(X_i)$ ∎

Anwendung der Theorie der Kontextfreien Sprachen

- Parsergeneratoren, beziehungsweise im Compilerbau
  1. Parsergenerator verwandelt die Beschreibung einer Sprache in einen Parser für diese Sprache
  2. Parser werden zur syntaktischen Analyse von Programmen verwendet
- Anwendungen im Bereich der Wissensrepräsentation, etwa in XML-Dokumenten (siehe Skriptum)

lexikalische Analyse

- der Eingabestrom aus `ASCII`-Zeichen wird analysiert
- in terminale Symbole der formalen Sprache zerlegt, die man Token nennt

Beispiel

Betrachte die Eingabe eines (simplen) Taschenrechners, dann sind die Token die Ziffern der Rechung, charakterisiert etwa durch den folgenden regulären Ausdruck

```
([0-9]+|([0-9]*\.[0-9]+))([eE][-+]?[0-9]+)?
```

mögliche Eingabe: $1.0 + (2.1e1 - 3 * 5) * 0.5$

die Regeln der lexikalischen Analyse werden in der folgenden Form definiert

Muster Aktion

Beispiel

KFG G_2 für arithmetische Ausdrücke

$$
\begin{aligned}
E &\to T \mid +T \mid -T \mid E + T \mid E - T \\
T &\to F \mid T \cdot F \mid T/F \\
F &\to \mathrm{c} \mid \mathrm{v} \mid (E)\ .
\end{aligned}
$$

in G_2 gilt etwa $-c * v \in L(E)$:

	Wort x	Variable	Regel	Rekursion
1	c	F	$F \to \mathrm{c}$	
2	v	F	$F \to \mathrm{v}$	
3	c	T	$T \to F$	1
4	$\mathrm{c} \cdot \mathrm{v}$	T	$T \to T \cdot F$	$3, 2$
5	$-\mathrm{c} \cdot \mathrm{v}$	E	$E \to -T$	4

Parsergenerierung in C mit yacc

Regelteil calc.y (gekürzt)

```
expression:     term                { $$ = $1; }
        |       '+' term            { $$ = $2; }
        |       '-' term            { $$ = -$2; }
        |       expression '+' term { $$ = $1 + $3; }
        |       expression '-' term { $$ = $1 - $3; }
        ;

term:           factor                     { $$ = $1; }
        |       term '*' factor            { $$ = $1 * $3; }
        |       term '/' factor            { $$ = $1 / $3; }
        ;

factor:         NUMBER                      { $$ = $1; }
        |       '(' expression ')'          { $$ = $2; }
        ;
```

Demo: Lex & Yacc