universität innsbruck

# Einführung in die Theoretische Informatik

David Drexel, Julian Fodor, Alexander Lochmann, **Georg Moser**, Alexander Maringele, David Obwaller, Jonas Schöpf

cbr.uibk.ac.at

universität innsbruck

# Zusammenfassung

## Zusammenfassung der letzten LVA

**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. $\Sigma$ 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: $\delta$ muss für alle möglichen Argumente definiert sein

**Satz**

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



**Satz (für KFG)**

*Gelte $A \Rightarrow X_1X_2 \dots X_n \overset{*}{\Rightarrow} x$, wobei $X_i \in V \cup \Sigma$, $x \in \Sigma^*$; dann können wir $x$ in die Stücke $x_1, x_2, \dots, x_n$ brechen, sodass für alle $i$: $X_i \overset{*}{\Rightarrow} x_i$*

**Beweis.**

1. Wenn $X_i \in \Sigma$, dann $X_i = x_i$ und offensichtlich $X_i \overset{*}{\Rightarrow} x_i$
2. Wenn $X_i \in V$, dann erhalten wir $X_i \overset{*}{\Rightarrow} x_i$ indem
   - Expansionen von $X_1, \dots, X_{i-1}$ eliminiert
   - Expansionen von $X_{i+1}, \dots, X_n$ eliminiert
   - überflüssige Schritte weggelassen werden
3. Beachte, dass Expansionen von $X_i$ immer links von Expansionen von $X_j$ sind, wenn $i < j$; das gilt aber nur für KFGs

■

## Einführung in die Logik

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

## Einführung in die Algebra

algebraische Strukturen, Beispiele von Algebren, Zusammenhang Boolesche Algebra und Aussagenlogik, Universelle Algebra, Satz von Birkhoff

## 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

Algorithmisch unlösbare Probleme, Turing Maschinen, Registermaschinen

## Einführung in die Programmverifikation

Prinzipien der Analyse von Programmen, Verifikation nach Hoare



## 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 $G_2 = (\{S\}, \{(,)\}, R, S)$, 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 $A \to x \in R$ mit $x \in (V \cup \Sigma)^*$
Die Bestimmung der Sprache $L(A)$ von $A$ mittels rekursiver Inferenz:

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

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

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

## 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` generiert einen bottom-up parser (nach der Pause)



## Beispiel

Wir betrachten die KFG $G_3 = (\{E, I\}, \{+, \cdot, (, ), a, b, 0, 1\}, R, E)$, wobei

$E \to I$ $\quad E \to E \cdot E$ $\quad I \to a$ $\quad I \to I0$

$E \to E+E$ $\quad E \to (E)$ $\quad I \to Ib$ $\quad I \to I1$

$\quad\quad\quad I \to Ia$ $\quad I \to b$

und zeigen $(a+b10) \in L(E)$:

| | Wort $x$ | Variable | Regel | Rekursion |
|---|---|---|---|---|
| 1 | a | $I$ | $I \to a$ | |
| 2 | b | $I$ | $I \to b$ | |
| 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 |

## 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 $\Sigma$
   - ein Nichtterminal aus $V$, oder
   - $\epsilon$

   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

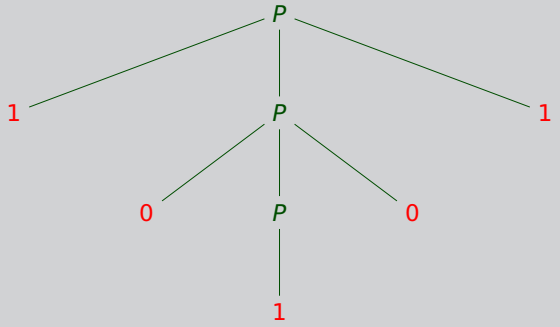


## Beispiel

Wir betrachten die KFG $G = (\{P\}, \Sigma, R, P)$ mit:

$$P \to \epsilon \mid 0 \mid 1 \mid 0P0 \mid 1P1$$

Dann ist der folgende Baum ein Syntaxbaum für $G_1$:

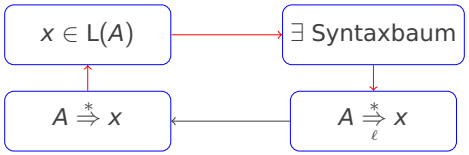





## Satz

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

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





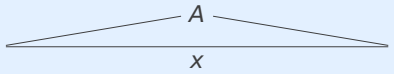

## Satz

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

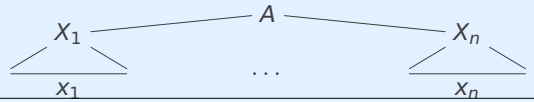
## Beweis.

Mit Induktion nach der Anzahl der Rekursionen im Inferenzverfahren

1. Basis $x \in L(A)$ benutzt genau die Regel $A \to x \in R$
   Dann $\exists$ Syntaxbaum mit Wurzel $A$:




2. Schritt $x \in L(A)$ benutzt $A \to X_1 \cdots X_n$ ($X_i \in V \cup \Sigma$)
   Dann $\exists$ Syntaxbaum mit Wurzel $A$:



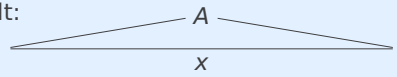
**Satz**

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

**Beweis.**

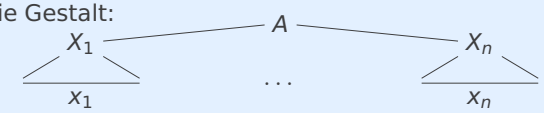
Mit Induktion nach der Höhe des Syntaxbaums

1. Basis Habe $S$ die Gestalt:




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

2. Schritt Habe $S$ 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 $A \overset{*}{\Rightarrow} x$ mit $x \in \Sigma^*$, dann liefert das rekursive Inferenzverfahren, dass $x \in \mathrm{L}(A)$*

**Beweis.**

Mit Induktion nach der Länge $\ell$ der Ableitung $A \overset{*}{\Rightarrow} x$:

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

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

   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 $X_i \overset{*}{\Rightarrow} x_i$ 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

## Lexikalische Analyse in C

### Regelteil calc.l

```
%%
([0-9]+|([0-9]*\.[0-9]+))([eE][-+]?[0-9]+)? {
  yylval.dval = atof(yytext);
  return NUMBER;
}
%%
```

### Parsergenerierung mit `lex` & `yacc`

Generell bestehen die Hilfsdateien des Parstergenerators aus

- einem Definitionsteil,
- einem Regelteil, der zwischen `%%` eingeschlossen wird, und
- einem Programmteil für benutzereigene Funktionen.



### Beispiel

Grammatik $G$ für arithmetische Ausdrücke

$$E \to T \mid +T \mid -T \mid E+T \mid E-T$$
$$T \to F \mid T*F \mid T/F$$
$$F \to \mathrm{c} \mid \mathrm{v} \mid (E)\,.$$

in $G$ 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 | $\mathrm{c}*\mathrm{v}$ | $T$ | $T \to T*F$ | 3, 2 |
| 5 | $-\mathrm{c}*\mathrm{v}$ | $E$ | $E \to -T$ | 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; }
        ;
```