Anwendungen kontextfreier Grammatiken

Vortrag im Rahmen des Proseminars Theoretische Informatik WS 10/11

von Jan Lottermoser

Ursprünglich waren kontextfreie Grammatiken als Mittel zur Beschreibung natürlicher Sprachen gedacht. Diese Erwartung hat sich jedoch nicht erfüllt. Erst als rekursive Konzepte in der Informatik stark zu nahmen, fanden kontextfreie Grammatiken erste Anwendungen.

Im Folgenden werden eine alte und eine neuere Anwendung kontextfreier Grammatiken vorgestellt:

1. Kontextfreie Grammatiken zur Beschreibung von Programmiersprachen (Parser)
2. Extensible Markup Language (XML) und Dokumenttypdefinitionen (DTD)

1. Beschreibung von Programmiersprachen (Parser)

Es gibt wichtige Aspekte von Programmiersprachen, die nicht durch reguläre Ausdrücke beschrieben werden können. Es folgen zwei Beispiele:

Beispiel a) L = Dyck-Sprache (Sprache der korrekten Klammerausdrücke)

G = ({B}, {(,)}, P, B) wobei P besteht aus:

$B \rightarrow BB \mid (B) \mid \varepsilon$

Behauptung: $L(G) = L$

Beweis: (1) $L(G) \subseteq L$

Offensichtlich erzeugt G nur Wörter aus L, da zwei Wörter aus L konkateniert oder ein Wort aus L umklammert wieder ein Wort aus L ergeben und ε auch in L liegt.

(2) $L \subseteq L(G)$

Induktion über Länge n eines Wortes w aus L. (n ist Anzahl der Klammerpaare)

IA: $n = 0: \; w = \varepsilon: \; B \underset{G}{\Rightarrow} \varepsilon$

$n = 1: \; w = (): \; B \underset{G}{\Rightarrow} (B) \underset{G}{\Rightarrow} ()$

IS: $n \geq 2: \; w = (u), \; |u| = n, \; u \in \{(,)\}^*$

1. Fall: $u \in L$

Nach Ind.Ann. gilt $B \underset{G}{\Rightarrow} u$ Somit ist Ableitung für w: $B \underset{G}{\Rightarrow} (B) \overset{*}{\underset{G}{\Rightarrow}} (u) = w$

2. Fall: $u \notin L \Rightarrow u = x)\, y(\, z, \Rightarrow x\, , y\, , z \in L$

Nach Ind.Ann. gilt: $B \overset{*}{\underset{G}{\Rightarrow}} x$ *und* $B \overset{*}{\underset{G}{\Rightarrow}} y$ *und* $B \overset{*}{\underset{G}{\Rightarrow}} z$ Somit ist die Ableitung für w:

$$B \underset{G}{\Rightarrow} BB \underset{G}{\Rightarrow} BBB \underset{G}{\Rightarrow} (B)BB \underset{G}{\Rightarrow} (B)B(B) \overset{*}{\underset{G}{\Rightarrow}} (x)y(z) = (x)y(z) = (u) = w \qquad \square$$

Beweis, dass L nicht regulär ist: (Pumping Lemma)

Wäre L regulär, so gäbe es ein $k \in \mathbb{N}$ für das sich jedes w aus L mit |w| ≥ k zerlegen lässt in

w = uvw mit |v| ≥ 1 und |uv| ≤ k , sodass gilt: $uv^i w \in L$ für i = 0, 1, 2 ...

Sei $w = ()^k$. Da |uv| ≤ k gilt, besteht v nur aus einer oder mehr öffnender Klammern. uw enthält somit mehr schließende als öffnende Klammern. □

Es gibt jedoch auch Strukturen in Programmiersprachen, die nicht immer ausgewogene Zeichenreihen enthalten. Ein Beispiel dafür ist die if-else-Struktur in C. Eine if-Anweisung kann mit oder ohne else-Klausel vorkommen.

Beispiel b) L = Sprache der korrekten if-else-Klauseln über dem Alphabet {i,e}

G = ({S},{i,e},P,{S}) wobei P aus den folgenden Produktionen besteht:

S → ε | SS | iS | iSe

Behauptung: $L(G) = L$

Beweis: (1) $L(G) \subseteq L$

Offensichtlich erzeugt G nur Wörter aus L: ε ist korrekte if-else-Klausel. Zwei konkatenierte korrekte if-else-Klauseln sind wieder korrekt. Vor eine korrekte if-else-Klausel kann man ein if stellen und sie bleibt korrekt und man kann eine korrekte if-else-Klausel mit i und e umranden, da das e sich entweder auf das erste i bezieht oder auf eines danach.

(2) $L \subseteq L(G)$

Induktion über die Länge n eines Wortes w aus L.

IA: $n = 0: \; w = \varepsilon: \; S \underset{G}{\Rightarrow} \varepsilon = w \qquad n = 1: \; w = i: \; S \underset{G}{\Rightarrow} iS \underset{G}{\Rightarrow} i = w$

IS: $n \geq 2: \; w = ixe$

1. Fall: e bezieht sich auf i. Dann muss gelten: $x \in L$ *und nach Ind.Ann.* $S \overset{*}{\underset{G}{\Rightarrow}} x$

Also: $S \underset{G}{\Rightarrow} iSe \overset{*}{\underset{G}{\Rightarrow}} ixe = w$

2. Fall: e bezieht sich auf ein i innerhalb von x. Somit hat x die Form x = uiv. Da das letzte e in w sich auf das i zwischen u und v bezieht, ist v aus L und nach Ind.Ann. $S \overset{*}{\underset{G}{\Rightarrow}} v$ Da w = iuive aus L und ive aus L muss auch iu aus L sein. Somit gilt:

$$S \underset{G}{\Rightarrow} SS \underset{G}{\Rightarrow} iSS \overset{*}{\underset{G}{\Rightarrow}} iuS \underset{G}{\Rightarrow} iuiSe \overset{*}{\underset{G}{\Rightarrow}} iuive = w \qquad \square$$

Der **Beweis**, dass L nicht regulär ist, ist analog zu dem aus Beispiel a) für das Wort $w = i^k e^k$. □

1.1 Der YACC-Parsergenerator

Ein Parser ist eine Funktion, die die Struktur eines Quelltextprogramms erkennt und sie durch einen Ableitungsbaum (Parsebaum) repräsentiert. Der YACC Befehl ist auf allen UNIX Systemen vorhanden und erzeugt aus gegebener kfG einen Parser.

2. Markupsprachen und XML

Markup Sprachen oder Auszeichnungssprachen sind Sprachen, die bestimmte Markierungen, genannt Tags, enthalten. Wörter dieser Sprachen werden Dokumente genannt und die Tags sagen etwas über die Semantik der verschiedenen Zeichenreihen innerhalb der Dokumente aus. Die bekannteste Auszeichnungssprache ist HTML und wie jede Programmiersprache kann sie durch eine kontextfreie Grammatik beschrieben werden. Einen Teil einer HTML Grammatik zeigt folgende Abbildung:

1. *Char* → a | A | …
2. *Text* → ε | *Char Text*
3. *Doc* → ε | *Element Doc*
4. *Element* → *Text* | <EM> *Doc* </EM> | <P> *Doc* | <OL> *List* </OL>
5. *ListItem* → <LI> *Doc*
6. *List* → ε | *ListItem List*

2.1 Extensible Markup Language (XML)

XML ist eine erweiterbare Auszeichungssprache. Dies bedeutet, dass man mithilfe von sogenannten Dokumenttypdefinitionen (DTD) die Sprache um Tags erweitern kann und dabei die zulässigen Strukturen innerhalb der Tags definiert.

Eine DTD ist im Grunde eine kontextfreie Grammatik mit spezieller Syntax. Eine DTD hat folgende Form:

<!DOCTYPE Name-der-DTD [`Liste der Elementdefinitionen`]>

Eine Elementdefinition sieht wiederum so aus:

```
<!ELEMENT Elementname (Beschreibung des Elements)>
```

Hier lässt sich schon erahnen, dass die Sprache zur Beschreibung von DTDs wieder durch eine kfG beschrieben werden kann.
Die Beschreibung eines Elements ist im Grunde ein regulärer Ausdruck. Grundlage dieser reg. Ausdrücke sind wieder Elemente und der spezielle Term #PCDATA, der für Text steht, der keine Tags beinhaltet. Die Operatoren sind Folgende:

1. | steht für die Vereinigung
2. Ein Komma ist die Konkatenation
3. * ist der gewöhnliche Sternoperator, + steht für ein oder mehr Vorkommen und ? steht für null oder ein Vorkommen

Die untere Abbildung zeigt ein Beispiel einer DTD für die Beschreibung von PCs.

```
<!DOCTYPE PcSpecs [
<!ELEMENT PCS          (PC*)>
<!ELEMENT PC           (MODEL, PRICE, PROCESSOR, RAM,DISK+)>
<!ELEMENT MODEL        (#PCDATA)>
<!ELEMENT PRICE        (#PCDATA)>
<!ELEMENT PROCESSOR    (MANF, MODEL, SPEED)>
<!ELEMENT MANF         (#PCDATA)>
<!ELEMENT MODEL        (#PCDATA)>
<!ELEMENT SPEED        (#PCDATA)>
<!ELEMENT RAM          (#PCDATA)>
<!ELEMENT DISK         (HARDDISK | CD | DVD)>
<!ELEMENT HARDDISK     (MANF, MODEL, SIZE)>
<!ELEMENT SIZE         (#PCDATA)>
<!ELEMENT CD           (SPEED)>
<!ELEMENT DVD          (SPEED)>
]>
```

Diese Notation für die Definition von DTDs entspricht nicht ganz der einer kfG, es gibt jedoch Regeln zur Umwandlung einer solchen kfG in eine gewöhnliche. Wir zeigen dies durch Induktion über die Größe des Ausdrucks um Produktionsrumpf.

Induktionsanfang:

Ist der Rumpf eine Verkettung von Elementen, so ist die Produktion schon in der gewöhnlichen kfG Notation:
`<!ELEMENT HARDDISK  (MANF, MODEL, SIZE)>` wird zu
Harddisk → Manf Model Size

Induktiosschritt:

Andernfalls sind abhängig vom letzten Operator im Produktionsrumpf fünf Fälle zu unterscheiden:

1. Die Produktion hat die Form $A \rightarrow E_1,E_2$, wobei E_1 und E_2 zulässige Ausdrücke sind. Wir führen zwei neue Variablen B und C ein, die sonst nirgends in der Grammatik vorkommen, und ersetzen die Produktion durch $A \rightarrow BC$, $B \rightarrow E_1$, $C \rightarrow E_2$.
   Die erste Produktion ist in gewöhnlicher Notation und da die Rümpfe der anderen beiden Produktionen kürzer sind als der Rumpf der ursprünglichen Produktion, kann man sie nach Induktionsannahme auch in die gewöhnliche Notation umwandeln. Dies gilt auch für alle folgenden Fälle.
2. Die Produktion hat die Form $A \rightarrow E_1 \mid E_2$. Sie wird ersetzt durch $A \rightarrow E_1$, $A \rightarrow E_2$.
3. Die Produktion hat die Form $A \rightarrow (E_1)^*$ und wird ersetzt durch $A \rightarrow BA$, $A \rightarrow \varepsilon$, $B \rightarrow E_1$.
4. Die Produktion hat die Form $A \rightarrow (E_1)^+$ und wird ersetzt durch $A \rightarrow BA$, $A \rightarrow B$, $B \rightarrow E_1$.
5. Die Produktion hat die Form $A \rightarrow (E_1)^?$ und wird ersetzt durch $A \rightarrow \varepsilon$, $A \rightarrow E_1$.