Lösungen zu den Übungsaufgaben im Buch *Automaten und Sprachen: Theoretische Informatik für die Praxis* von Andreas Müller, ISBN 978-3-662-70145-4 (Softcover), ISBN 978-3-662-70146-1 (eBook), https://link.springer.com/book/10.1007/978-3-662-70146-1. Website zum Buch: https://autospr.ch

Kapitel 6: Parsing

6.1. LDAP-Filter werden verwendet, um aus einem LDAP-Verzeichnis eine Menge von Knoten aufgrund von Werten einzelner Attribute auszuwählen. Elementare LDAP-Suchfilter haben die Form

`attribut=wert`

Sie selektieren genau die Knoten, deren Attribut `attribut` den Wert `wert` hat. Für die Zwecke dieser Aufgabe sind Attribute und Werte nichtleere Strings aus Buchstaben und Ziffern. Daraus lassen sich mithilfe von Klammern und logischen Verknüpfungen beliebige LDAP-Filter aufbauen. Die logischen Verknüpfungen verwenden eine Präfix-Notation:

UND:	`(&(attr1=wert1)(attr2=wert2))`
ODER:	`(|(attr1=wert1)(attr2=wert2))`
NICHT:	`(!(attr=wert))`

Bei der UND- und der ODER-Verknüpfung dürfen beliebig viele Filter miteinander verknüpft werden.

a) Gibt es einen regulären Ausdruck, der LDAP-Filter akzeptiert?

b) Bilden die syntaktisch korrekten LDAP-Filter eine kontextfreie Sprache?

Lösung. a) Dazu müsste die Sprache der LDAP-Filter regulär sein. Da LDAP-Filter beliebig tief geschachtelte Klammern verwenden dürfen, kann man mit dem Pumping-Lemma wie in Aufgabe **??** zeigen, dass die Sprache nicht regulär ist.

b) Die Sprache der LDAP-Filter ist kontextfrei, denn man kann dafür eine Grammatik aufstellen.

$$
\begin{aligned}
\text{filter} &\to \text{elementaryfilter}\\
&\to \text{pfilter}\\
\text{elementaryfilter} &\to \text{string '=' string}\\
\text{pfilter} &\to \text{'(' elementaryfilter ')'}\\
&\to \text{'(' pfilter ')'}\\
&\to \text{'(' '\&' pfiltersequence ')'}\\
&\to \text{'(' '|' pfiltersequence ')'}\\
&\to \text{'(' '!' pfilter ')'}\\
\text{pfiltersequence} &\to \text{pfilter}\\
&\to \text{pfiltersequence pfilter}\\
\text{string} &\to \text{char}\\
&\to \text{string char}\\
\text{char} &\to \text{'a' | 'b' | 'c' | ... | 'z' | '0' | ... | '9'}
\end{aligned}
$$

○

6.2. Die Sprache Lisp hat eine etwas eigene Syntax, Larry Wall hat sie einst mit "Haferbrei vermischt mit abgeschnittenen Fingernägeln" verglichen. In Lisp gibt es Atome:

- numerische Atome, d. h. Zahlen: `1 2 3 -4 3.14 -7.5 6.02E+23`
- Strings, also Zeichenfolgen in Anführungszeichen:

  `"ein String" "&]$787?" "setq" "12"`
- Symbole, alle anderen Zeichenketten, die von Zahlen und Strings unterscheidbar sind und kein Klammern und Anführungszeichen enthalten.

Aus den Atomen können jetzt Listen kombiniert werden. Eine Liste besteht aus einer öffnenden Klammer, einer Folge von Listenelementen und einer schließenden Klammer. Listenelemente sind entweder Atome oder Listen. Listen können auch leer sein, ebenso Strings. Ein typisches Lisp-Programm ist

```
(defun print-quadratzahlen (x)
  (cond ((plusp x)
         (print (* x x))
         (print-quadratzahlen (- x 1)))))
```

a) Formulieren Sie eine kontextfreie Grammatik für Lisp.

b) Falls Ihre Grammatik nicht Chomsky-Normalform hat: markieren Sie alle Regeln, die der Normalform widersprechen.

Hinweis. Zur Vereinfachung gehen Sie davon aus, dass Zahlen positive natürliche Zahlen sind, dass Strings nur Kleinbuchstaben und Ziffern enthalten, und dass ein Symbol eine Folge von Kleinbuchstaben (keine Ziffern) ist.

Lösung. a) Die folgende Grammatik folgt ziemlich genau der obigen Beschreibung der Sprache. Man beachte, dass die beiden ε-Regeln erlauben, dass Listen und Strings leer sind.

$$
\begin{aligned}
\text{liste} &\to \text{'('} \ \text{listenelemente} \ \text{')'} \\
\text{listenelemente} &\to \varepsilon \mid \text{listenelemente element} \\
\text{element} &\to \text{liste} \mid \text{atom} \\
\text{atom} &\to \text{zahl} \mid \text{string} \mid \text{symbol} \\
\text{zahl} &\to \text{ziffer} \mid \text{zahl ziffer} \\
\text{symbol} &\to \text{buchstabe} \mid \text{symbol buchstabe} \\
\text{string} &\to \text{'"'} \ \text{stringinhalt} \ \text{'"'} \\
\text{stringinhalt} &\to \varepsilon \mid \text{stringinhalt zeichen} \\
\text{ziffer} &\to \text{'0'} \mid \text{'1'} \mid \text{'2'} \mid \ldots \mid \text{'9'} \\
\text{buchstabe} &\to \text{'a'} \mid \text{'b'} \mid \ldots \mid \text{'z'} \\
\text{zeichen} &\to \text{ziffer} \mid \text{buchstabe}
\end{aligned}
$$

b) In der Grammatik oben sind Unit rules rot, Regeln mit mehr als zwei Symbolen auf der rechten Seite blau, und ε-Regeln, die nicht von der Startvariablen ausgehen, grün markiert. ○

6.3. Verwenden Sie die Expression-Term-Factor-Grammatik von Abschnitt 5.3.1 und den Cocke-Younger-Kasami-Algorithmus, um den Syntaxbaum des Ausdrucks `7*(5+3)` zu ermitteln.

Lösung. Wir verwenden die Expression-Term-Factor-Grammatik in der folgenden Form,

$$
\begin{aligned}
E &\to E + T \mid T \\
T &\to T * F \mid F \\
F &\to N \mid \texttt{(}\, E \,\texttt{)} \\
N &\to NZ \mid Z \\
Z &\to \texttt{0} \mid \texttt{1} \mid \cdots \mid \texttt{9}
\end{aligned}
$$

Für die Anwendung des Cocke-Younger-Kassami-Algorithmus wird die Grammatik in Chomsky-Normalform benötigt. Es ist aber nicht unbedingt nötig, die volle Chomsky-Normalform zu erreichen, zum Beispiel ist akzeptabel, wenn die Startvariable auf der rechten Seite vorkommt. Auf jeden Fall darf es aber keine Unit-Rules mehr geben und auch keine Dreierregeln wie $E \to E{+}T$, wir führen daher nur diese Schritt des Algorithmus zur Gewinnung der Chomsky-Normalform durch.

In einem Parser würde man die Analyse der Zahlen (Variable N) von einem Tokenizer durchführen lassen, der ganze Zahlen liefert. Wir betrachten daher der Einfachheit halber N als ein Terminalsymbol und lassen die Regeln mit Z weg, wir beginnen also mit den Regeln

$$
\begin{aligned}
E &\to E + T \mid T \\
T &\to T * F \mid F \\
F &\to N \mid \texttt{(}\, E \,\texttt{)}
\end{aligned}
$$

Die letzte Regel $F \to N$ wie eine Unit-Rule aussieht, schreiben wir `N` statt N, was den Regeln die Form

$$
\begin{aligned}
E &\to E + T \mid T \\
T &\to T * F \mid F \\
F &\to \texttt{N} \mid \texttt{(}\, E \,\texttt{)}
\end{aligned}
$$

gibt.

Wir eliminieren jetzt die Unit-Rules. Eliminiation von $T \to F$ ergibt

$$
\begin{aligned}
E &\to E + T \mid T \\
T &\to T * F \mid \texttt{N} \mid \texttt{(}\, E \,\texttt{)} \\
F &\to \texttt{N} \mid \texttt{(}\, E \,\texttt{)}
\end{aligned}
$$

Elimination von $E \to T$ ergibt

$$
\begin{aligned}
E &\to E + T \mid T * F \mid \texttt{N} \mid \texttt{(}\, E \,\texttt{)} \\
T &\to T * F \mid F \\
F &\to \texttt{N} \mid \texttt{(}\, E \,\texttt{)}
\end{aligned}
$$

Eliminiation von $T \to F$:

$$
E \to E + T \mid T * F \mid \texttt{N} \mid \texttt{(}\, E \,\texttt{)}
$$

$$T \to T \texttt{ * } F \mid \texttt{N} \mid \texttt{(} E \texttt{)}$$
$$F \to \texttt{N} \mid \texttt{(} E \texttt{)}$$

Damit sind alle Unit-Rules entfernt.

Die verbleibenden Terminalsymbole können durch die Regeln

$$P \to \texttt{+}$$
$$M \to \texttt{*}$$
$$O \to \texttt{(}$$
$$C \to \texttt{)}$$

aus den Regeln entfernt werden:

$$E \to E\ P\ T \mid T\ M\ F \mid \texttt{N} \mid O\ E\ C$$
$$T \to T\ M\ F \mid \texttt{N} \mid O\ E\ C$$
$$F \to \texttt{N} \mid O\ E\ C$$
$$P \to \texttt{+}$$
$$M \to \texttt{*}$$
$$O \to \texttt{(}$$
$$C \to \texttt{)}$$

Im letzten Schritt müssen jetzt die Dreiergruppen auf der rechten Seite aufgespaltet werden:

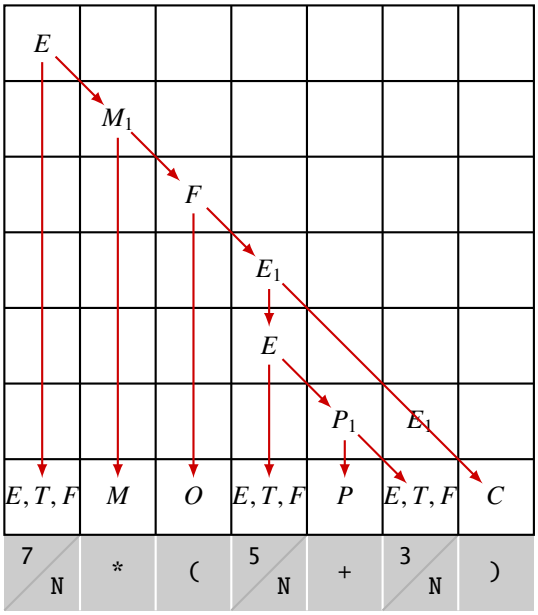
$$E \to E\ P_1 \mid T\ M_1 \mid \texttt{N} \mid O\ E_1$$
$$T \to T\ M_1 \mid \texttt{N} \mid O\ E_1$$
$$F \to \texttt{N} \mid O\ E_1$$
$$P \to \texttt{+}$$
$$M \to \texttt{*}$$
$$O \to \texttt{(}$$
$$C \to \texttt{)}$$
$$P_1 \to P\ T$$
$$M_1 \to M\ F$$
$$E_1 \to E\ C$$

Damit ist für die Zwecke der Anwendung im Cocke-Younger-Kassami-Algorithmus eine ausreichende Approximation der Chomsky-Normalform erreicht.

Mit dieser Grammatik kann jetzt die Cocke-Younger-Kassami-Tabelle ausgefüllt werden. Dies beginnt mit der Auflösung der Terminalsymbole in Variablen in der untersten Zeile. Dabei ist zu beachten, dass für die Zahlen die drei möglichen Variablen E, T und F gleichberechtigt nebeneinander in die entsprechenden Felder der untersten Tabellenzeile eingetragen werden müssen.

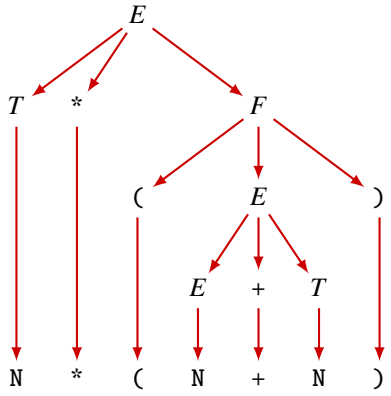
Die verbleibenden Einträge können dann mit den Regeln ergänzt werden, es ergibt sich die Ta-

belle



(1)

Aus der Tabelle (1) kann auch der Syntaxbaum abgelesen werden, der natürlich die für die Chomsky-Normalform nötigen Variablen M_1, E_1 und P_1 enthält. Löst man diese wieder auf, ergbit sich der Syntaxbaum



in der vertrauten Form. ○