Lösungen zu den Aufgaben zur Klausurvorbereitung

Aufgabe 1: Die Grammatik $G = \langle \{S\}, \{\ \text{“+”}\ ,\ \text{“*”}\ ,\ \text{“a”}\ \}, R, S \rangle$ habe die folgenden Regeln:

$$S \to S\ S\ \text{“+”}\ \mid S\ S\ \text{“-”}\ \mid\ \text{“a”}\ .$$

(a) Berechnen Sie die Mengen $\mathit{First}(S)$ und $\mathit{Follow}(S)$.

(b) Berechnen Sie die Menge der SLR-Zustände für diese Grammatik.

(c) Berechnen Sie die Funktionen *action*() und *goto*() für diese Grammatik.

(d) Berechnen Sie die Menge der LR-Zustände für diese Grammatik.

Lösung:

(a) Es gilt offenbar

$$\mathit{First}(S) = \{a\} \quad \text{und} \quad \mathit{Follow}(S) = \{\ \text{“+”}\ ,\ \text{“-”}\ ,\ \text{“a”}\ , \$\}.$$

(b) Wir erhalten die folgenden Zustände:

(a) Wir definieren $s_0 = \mathit{closure}(\{\widehat{S} \to \star S\})$ und finden

$$s_0 = \{\ \widehat{S} \to \star\, S,\ S \to \star\ \text{“a”}\ ,\ S \to \star\, S\ S\ \text{“+”}\ ,\ S \to \star\, S\ S\ \text{“-”}\ \}.$$

(b) Wir definieren $s_1 = \mathit{goto}(s_0, S)$ und finden

$$\begin{array}{rcll} s_1 & = & \{ & \widehat{S} \to S \star, \\ & & & S \to \star\ \text{“a”}\ , \\ & & & S \to \star\, S\ S\ \text{“+”}\ , \\ & & & S \to \star\, S\ S\ \text{“-”}\ , \\ & & & S \to S \star S\ \text{“+”}\ , \\ & & & S \to S \star S\ \text{“-”} \\ & & \}. & \end{array}$$

(c) Wir definieren $s_2 = \mathit{goto}(s_1, S)$ und finden

$$\begin{array}{rcll} s_2 & = & \{ & S \to \star\ \text{“a”}\ , \\ & & & S \to \star\, S\ S\ \text{“+”}\ , \\ & & & S \to \star\, S\ S\ \text{“-”}\ , \\ & & & S \to S \star S\ \text{“+”}\ , \\ & & & S \to S \star S\ \text{“-”}\ , \\ & & & S \to S\ S \star\ \text{“+”}\ , \\ & & & S \to S\ S \star\ \text{“-”} \\ & & \}. & \end{array}$$

(d) Wir definieren $s_3 = \mathit{goto}(s_2,\ \text{“a”}\ )$ und finden

$$s_3 = \{S \to\ \text{“a”}\ \star\}.$$

(e) Wir definieren $s_4 = \mathit{goto}(s_2,\ \text{“+”}\ )$ und finden

$$s_4 = \{S \to S\ S\ \text{“+”}\ \star\}.$$

(f) Wir definieren $s_5 = \mathit{goto}(s_2,\ \text{“-”}\ )$ und finden

$$s_5 = \{S \to S\ S\ \text{“-”}\ \star\}.$$

(c) Damit erhalten wir für die Funktion *action*() die folgende Tabelle:

(a) $action(s_0,\ \text{“a”}\ ) = \langle \texttt{shift}, s_3 \rangle$
(b) $action(s_1,\ \text{“\$”}\ ) = \texttt{accept}$
(c) $action(s_1,\ \text{“a”}\ ) = \langle \texttt{shift}, s_3 \rangle$
(d) $action(s_2,\ \text{“+”}\ ) = \langle \texttt{shift}, s_4 \rangle$
(e) $action(s_2,\ \text{“-”}\ ) = \langle \texttt{shift}, s_5 \rangle$
(f) $action(s_2,\ \text{“a”}\ ) = \langle \texttt{shift}, s_3 \rangle$
(g) $action(s_3,\ \text{“\$”}\ ) = \langle \texttt{reduce}, S \rightarrow\ \text{“a”}\ \rangle$
(h) $action(s_3,\ \text{“+”}\ ) = \langle \texttt{reduce}, S \rightarrow\ \text{“a”}\ \rangle$
(i) $action(s_3,\ \text{“-”}\ ) = \langle \texttt{reduce}, S \rightarrow\ \text{“a”}\ \rangle$
(j) $action(s_3,\ \text{“a”}\ ) = \langle \texttt{reduce}, S \rightarrow\ \text{“a”}\ \rangle$
(k) $action(s_4,\ \text{“\$”}\ ) = \langle \texttt{reduce}, S \rightarrow S\ S\ \text{“+”}\ \rangle$
(l) $action(s_4,\ \text{“+”}\ ) = \langle \texttt{reduce}, S \rightarrow S\ S\ \text{“+”}\ \rangle$
(m) $action(s_4,\ \text{“-”}\ ) = \langle \texttt{reduce}, S \rightarrow S\ S\ \text{“+”}\ \rangle$
(n) $action(s_4,\ \text{“a”}\ ) = \langle \texttt{reduce}, S \rightarrow S\ S\ \text{“+”}\ \rangle$
(o) $action(s_5,\ \text{“\$”}\ ) = \langle \texttt{reduce}, S \rightarrow S\ S\ \text{“-”}\ \rangle$
(p) $action(s_5,\ \text{“+”}\ ) = \langle \texttt{reduce}, S \rightarrow S\ S\ \text{“-”}\ \rangle$
(q) $action(s_5,\ \text{“-”}\ ) = \langle \texttt{reduce}, S \rightarrow S\ S\ \text{“-”}\ \rangle$
(r) $action(s_5,\ \text{“a”}\ ) = \langle \texttt{reduce}, S \rightarrow S\ S\ \text{“-”}\ \rangle$

Für die Funktion *goto*() finden wir:

(a) $goto(s_0, S) = s_1$
(b) $goto(s_1, S) = s_2$
(c) $goto(s_2, S) = s_2$

(d) Wir erhalten die folgenden Zustände:

(a) Wir setzen wieder $s_0 = closure\Big(\big\{\widehat{S} \rightarrow \star S : \$\big\}\Big)$ und erhalten diesmal

$$\begin{aligned} s_0 \ = \ \{ \quad & \widehat{S} \rightarrow \star\, S :\ \text{“\$”}\ , \\ & S \rightarrow \star\ \text{“a”}\ \ :\{\ \text{“\$”}\ ,\ \text{“a”}\ \}, \\ & S \rightarrow \star\, S\ S\ \text{“+”}\ \ :\{\ \text{“\$”}\ ,\ \text{“a”}\ \}, \\ & S \rightarrow \star\, S\ S\ \text{“-”}\ \ :\{\ \text{“\$”}\ ,\ \text{“a”}\ \} \\ \}. & \end{aligned}$$

(b) Wir definieren $s_1 = goto(s_0, S)$ und erhalten

$$\begin{aligned} s_1 \ = \ \{ \quad & \widehat{S} \rightarrow S\star :\ \text{“\$”}\ , \\ & S \rightarrow \star\ \text{“a”}\ \ :\{\ \text{“+”}\ ,\ \text{“-”}\ ,\ \text{“a”}\ \}, \\ & S \rightarrow \star\, S\ S\ \text{“+”}\ \ :\{\ \text{“+”}\ ,\ \text{“-”}\ ,\ \text{“a”}\ \}, \\ & S \rightarrow \star\, S\ S\ \text{“-”}\ \ :\{\ \text{“+”}\ ,\ \text{“-”}\ ,\ \text{“a”}\ \}, \\ & S \rightarrow S\star\, S\ \text{“+”}\ \ :\{\ \text{“\$”}\ ,\ \text{“a”}\ \}, \\ & S \rightarrow S\star\, S\ \text{“-”}\ \ :\{\ \text{“\$”}\ ,\ \text{“a”}\ \} \\ \}. & \end{aligned}$$

(c) Wir definieren $s_2 = goto(s_1, S)$ und erhalten

$$\begin{aligned} s_2 \;=\; \{\quad & S \to \star\ \text{“a”} \;:\{\ \text{“+”}\ ,\ \text{“-”}\ ,\ \text{“a”}\ \},\\ & S \to \star\ S\ S\ \text{“+”} \;:\{\ \text{“+”}\ ,\ \text{“-”}\ ,\ \text{“a”}\ \},\\ & S \to \star\ S\ S\ \text{“-”} \;:\{\ \text{“+”}\ ,\ \text{“-”}\ ,\ \text{“a”}\ \},\\ & S \to S \star S\ \text{“+”} \;:\{\ \text{“+”}\ ,\ \text{“-”}\ ,\ \text{“a”}\ \},\\ & S \to S \star S\ \text{“-”} \;:\{\ \text{“+”}\ ,\ \text{“-”}\ ,\ \text{“a”}\ \},\\ & S \to S\ S \star\ \text{“+”} \;:\{\ \text{“\$”}\ ,\ \text{“a”}\ \},\\ & S \to S\ S \star\ \text{“-”} \;:\{\ \text{“\$”}\ ,\ \text{“a”}\ \}\\ \}. \end{aligned}$$

(d) Wir definieren $s_3 = goto(s_2, S)$ und erhalten

$$\begin{aligned} s_3 \;=\; \{\quad & S \to \star\ \text{“a”} \;:\{\ \text{“+”}\ ,\ \text{“-”}\ ,\ \text{“a”}\ \},\\ & S \to \star\ S\ S\ \text{“+”} \;:\{\ \text{“+”}\ ,\ \text{“-”}\ ,\ \text{“a”}\ \},\\ & S \to \star\ S\ S\ \text{“-”} \;:\{\ \text{“+”}\ ,\ \text{“-”}\ ,\ \text{“a”}\ \},\\ & S \to S \star S\ \text{“+”} \;:\{\ \text{“+”}\ ,\ \text{“-”}\ ,\ \text{“a”}\ \},\\ & S \to S \star S\ \text{“-”} \;:\{\ \text{“+”}\ ,\ \text{“-”}\ ,\ \text{“a”}\ \},\\ & S \to S\ S \star\ \text{“+”} \;:\{\ \text{“+”}\ ,\ \text{“-”}\ ,\ \text{“a”}\ \},\\ & S \to S\ S \star\ \text{“-”} \;:\{\ \text{“+”}\ ,\ \text{“-”}\ ,\ \text{“a”}\ \}\\ \}. \end{aligned}$$

(e) Wir definieren $s_4 = goto(s_0,\ \text{“a”}\ )$ und erhalten

$$s_4 = \{S \to\ \text{“a”}\ \star : \{\ \text{“\$”}\ ,\ \text{“a”}\ \}\}.$$

(f) Wir definieren $s_5 = goto(s_2,\ \text{“a”}\ )$ und erhalten

$$s_5 = \big\{S \to\ \text{“a”}\ \star : \{\ \text{“+”}\ ,\ \text{“-”}\ ,\ \text{“a”}\ \}\big\}$$

(g) Wir definieren $s_6 = goto(s_2,\ \text{“+”}\ )$ und erhalten

$$s_6 = \big\{S \to S\ S\ \text{“+”}\ \star : \{\ \text{“\$”}\ ,\ \text{“a”}\ \}\big\}$$

(h) Wir definieren $s_7 = goto(s_3,\ \text{“+”}\ )$ und erhalten

$$s_7 = \big\{S \to S\ S\ \text{“+”}\ \star : \{\ \text{“+”}\ ,\ \text{“-”}\ ,\ \text{“a”}\ \}\big\}$$

(i) Wir definieren $s_8 = goto(s_2,\ \text{“-”}\ )$ und erhalten

$$s_8 = \big\{S \to S\ S\ \text{“-”}\ \star : \{\ \text{“\$”}\ ,\ \text{“a”}\ \}\big\}$$

(j) Wir definieren $s_9 = goto(s_3,\ \text{“-”}\ )$ und erhalten

$$s_9 = \big\{S \to S\ S\ \text{“-”}\ \star : \{\ \text{“+”}\ ,\ \text{“-”}\ ,\ \text{“a”}\ \}\big\}$$

Aufgabe 2: Die Grammatik $G = \langle \{A, B\}, \{ \text{“u”} , \text{“x”} , \text{“y”} , \text{“z”} \}, R, A \rangle$ habe die folgenden Regeln:

$$\begin{array}{lcl} A & \rightarrow & B \text{ “x”} \\ & | & \text{“y”} \; B \text{ “z”} \\ & | & \text{“u”} \;\; \text{“z”} \\ & | & \text{“y”} \;\; \text{“u”} \;\; \text{“x”} \\ B & \rightarrow & \text{“u”} \end{array}$$

Bearbeiten Sie die folgenden Teilaufgaben:

(a) Überprüfen Sie, ob die diese Grammatik eine LL(1)-Grammatik ist und begründen Sie Ihre Antwort.

(b) Überprüfen Sie, ob die diese Grammatik eine LL(∗)-Grammatik ist und begründen Sie Ihre Antwort.

(c) Überprüfen Sie, ob die diese Grammatik eine SLR-Grammatik ist und begründen Sie Ihre Antwort.

Lösung:

(a) Die Grammatik ist keine $LL(1)$-Grammatik, denn zwischen den beiden Regeln

$$A \rightarrow \text{“y”} \; B \text{ “z”} \quad \text{und} \quad A \rightarrow \text{“y”} \;\; \text{“u”} \;\; \text{“x”}$$

gibt es einen Konflikt, wir haben

$$\mathit{First}(\text{“y”} \; B \text{ “z”}) = \{ \text{“y”} \} \quad \text{und} \quad \mathit{First}(\text{“y”} \;\; \text{“u”} \;\; \text{“z”}) = \{ \text{“y”} \}$$

und damit folgt

$$\mathit{First}(\text{“y”} \; B \text{ “z”}) \cap \mathit{First}(\text{“y”} \;\; \text{“u”} \;\; \text{“z”}) = \{ \text{“y”} \} \neq \{\}.$$

(b) Um zu überprüfen, ob die Grammatik eine LL(∗)-Grammatik ist, annotieren wir die Regeln wie im Skript beschrieben mit Zuständen und erhalten die folgenden annotierten Regeln:

(a) $A \rightarrow_{\langle 0,1 \rangle} B_{\langle 1,1 \rangle} \text{ “x”}_{\langle 2,1 \rangle}$
(b) $A \rightarrow_{\langle 0,2 \rangle} \text{ “y”}_{\langle 1,2 \rangle} B_{\langle 2,2 \rangle} \text{ “z”}_{\langle 3,2 \rangle}$
(c) $A \rightarrow_{\langle 0,3 \rangle} \text{ “u”}_{\langle 1,3 \rangle} \text{ “z”}_{\langle 2,3 \rangle}$
(d) $A \rightarrow_{\langle 0,4 \rangle} \text{ “y”}_{\langle 1,4 \rangle} \text{ “u”}_{\langle 2,4 \rangle} \text{ “x”}_{\langle 3,4 \rangle}$
(e) $B \rightarrow_{\langle 5,0 \rangle} \text{ “u”}_{\langle 6,0 \rangle}$

Als Start-Zustand definieren wir den Zustand $\langle 0, 0 \rangle$ und setzen $\mathit{start}(B) = \langle 4, 0 \rangle$ sowie $\mathit{end}(B) = \langle 7, 0 \rangle$. Die Zustands-Übergangs-Funktion des für einen LL(∗)-Parser zu konstruierenden nicht-deterministischen Automaten ist dann wie folgt gegeben:

(a) Der Start-Zustand ist mit den Anfangs-Zuständen der einzelnen Regeln für A verbunden, wir haben also

$$\delta(\langle 0, 0 \rangle, \varepsilon) = \langle 0, 1 \rangle, \quad \delta(\langle 0, 0 \rangle, \varepsilon) = \langle 0, 2 \rangle,$$
$$\delta(\langle 0, 0 \rangle, \varepsilon) = \langle 0, 3 \rangle, \quad \delta(\langle 0, 0 \rangle, \varepsilon) = \langle 0, 4 \rangle.$$

(b) Aus der Regel $A \rightarrow B \text{ “x”}$ erhalten wir die Übergänge

$$\delta(\langle 0, 1 \rangle, \varepsilon) = \langle 4, 0 \rangle, \quad \delta(\langle 7, 0 \rangle, \varepsilon) = \langle 1, 1 \rangle, \quad \delta(\langle 1, 1 \rangle, \text{ “x”}) = \langle 2, 1 \rangle.$$

(c) Aus der Regel $A \rightarrow$ "y" B "z" erhalten wir die Übergänge

$$\delta(\langle 0,2\rangle,\ \text{“y”}\ ) = \langle 1,2\rangle, \quad \delta(\langle 1,2\rangle, \varepsilon) = \langle 4,0\rangle,$$
$$\delta(\langle 7,0\rangle, \varepsilon) = \langle 2,2\rangle, \quad \delta(\langle 2,2\rangle,\ \text{“z”}\ ) = \langle 3,2\rangle.$$

(d) Aus der Regel $A \rightarrow$ "u" "z" erhalten wir die Übergänge

$$\delta(\langle 0,3\rangle,\ \text{“u”}\ ) = \langle 1,3\rangle, \quad \delta(\langle 1,3\rangle,\ \text{“z”}\ ) = \langle 2,3\rangle.$$

(e) Aus der Regel $A \rightarrow$ "y" "u" "x" erhalten wir die Übergänge

$$\delta(\langle 0,4\rangle,\ \text{“y”}\ ) = \langle 1,4\rangle, \quad \delta(\langle 1,4\rangle,\ \text{“u”}\ ) = \langle 2,4\rangle, \quad \delta(\langle 2,4\rangle,\ \text{“x”}\ ) = \langle 3,4\rangle.$$

(f) Aus der Regel $B \rightarrow$ "u" erhalten wir die Übergänge

$$\delta(\langle 4,0\rangle, \varepsilon) = \langle 5,0\rangle, \quad \delta(\langle 5,0\rangle,\ \text{“u”}\ ) = \langle 6,0\rangle, \quad \delta(\langle 6,0\rangle, \varepsilon) = \langle 7,0\rangle.$$

(g) Die Menge der akzeptierenden Zustände ist durch die Endzustände der Regeln für die Variable A gegeben:

$$\big\{\langle 2,1\rangle, \langle 3,2\rangle, \langle 2,3\rangle, \langle 3,4\rangle\big\}$$

Wir beginnen nun damit, diesen nicht-deterministischen Automaten in einen deterministischen Automaten zu überführen. Der Start-Zustand des deterministischen Automaten ist dann

$$S_0 := \big\{\langle 0,0\rangle, \langle 0,1\rangle, \langle 0,2\rangle, \langle 0,3\rangle, \langle 0,4\rangle, \langle 4,0\rangle, \langle 5,0\rangle\big\}.$$

Wir definieren

$$S_1 := \Delta(S_0,\ \text{“u”}\ ) = \big\{\langle 6,0\rangle, \langle 7,0\rangle, \langle 1,1\rangle, \langle 2,2\rangle, \langle 1,3\rangle\big\}.$$

Weiter definieren wir

$$S_2 := \Delta(S_1,\ \text{“z”}\ ) = \big\{\langle 3,2\rangle, \langle 2,3\rangle\big\}.$$

Diese Menge enthält zwei akzeptierende Zustände, ist aber nicht homogen, da diese Zustände einen unterschiedlichen Index haben. Folglich handelt es sich bei der obigen Grammatik nicht um eine LL($*$)-Grammatik.

Bemerkung: ANTLR kann aus der obigen Grammatik trotzdem einen Parser erzeugen, weil ANTLR die Regel $B \rightarrow$ "u" expandiert und dann die folgende Grammatik analysiert:

$$\begin{array}{lcl} A & \rightarrow & \text{“u”}\ \ \text{“x”} \\ & | & \text{“y”}\ \ \text{“u”}\ \ \text{“z”} \\ & | & \text{“u”}\ \ \text{“z”} \\ & | & \text{“y”}\ \ \text{“u”}\ \ \text{“x”} \end{array}$$

Die so umgeformte Grammatik ist eine LL($*$)-Grammatik.

(c) Die angegebene Grammatik ist keine SLR-Grammatik. Um das zu sehen, erweitern wir die Grammatik um die Regel $\widehat{S} \rightarrow A$ und berechnen den Zustand

$$s_0 = \mathit{closure}(\{\widehat{S} \rightarrow A\}).$$

Wir finden

$$\begin{array}{rcll} s_0 & = & \{ & S \rightarrow \star\, A, \\ & & & A \rightarrow \star\, B\ \text{“x”}\ , \\ & & & A \rightarrow \star\ \text{“y”}\ B\ \text{“z”}\ , \\ & & & A \rightarrow \star\ \text{“u”}\ \ \text{“z”}\ , \\ & & & A \rightarrow \star\ \text{“y”}\ \ \text{“u”}\ \ \text{“x”}\ , \\ & & & B \rightarrow \star\ \text{“u”} \\ & & \}. & \end{array}$$

Wir berechnen nun $\mathit{goto}(s_0,\ \text{“u”}\ )$ und erhalten

$s_1 = \{A \rightarrow$ "u" $\star$ "z" , $B \rightarrow$ "u" $\star\}$.

Bei der Berechnung von *action*(s_1, "z") tritt nun eine Shift-Reduce-Konflikt auf, denn es gilt

follow(B) = { "x" , "z" }.

Bemerkung: Die in der Aufgabe angegebene Grammatik ist sowohl eine LR-Grammatik als auch eine LALR-Grammatik. Letzteres läßt sich mit *Bison* oder *JavaCup* nachweisen und Ersteres folgt aus der Tatsache, dass jede LALR-Grammatik auch eine LR-Grammatik ist.

Aufgabe 3: Wir definieren *geschachtelte Listen* rekursiv als solche Listen, deren Elemente natürliche Zahlen oder geschachtelte Listen sind. Die Elemente in geschachtelten Listen sollen durch Kommata getrennt werden und die Listen selber sollen durch die eckigen Klammern "`[`" und "`]`" begrenzt sein. Beispiele für geschachtelte Listen sind also:

(a) `[ 1, [ 1, [], [ 2, 3]], 7]`

(b) `[ [], [ [], [], [ 4], 5], []]`

Lösen Sie die folgenden Teilaufgaben:

(a) Geben Sie eine Grammatik für geschachtelte Listen an.

(b) Geben Sie die EP-Spezifikation von Klassen an, mit deren Hilfe sich geschachtelte Listen repräsentieren lassen.

(c) Geben Sie einen *JavaCup*-Parser an, der eine geschachtelte Liste einliest und einen abstrakten Syntax-Baum der Liste berechnet.

Hinweis: In der Klausur können Sie später davon ausgehen, dass ein geeigneter *JFlex*-Scanner bereits gegeben ist, aber bei dieser Aufgabe sollen Sie den Scanner ebenfalls erstellen, damit Sie Ihre Lösung auch testen können.

Lösung:

(a) Die Grammatik $G = \langle \{S, L, N, E\}, \{ \text{“[”}, \text{“]”}, \text{“,”}\ \textsc{Number}\}, R, S\rangle$, deren Regeln durch

$$
\begin{array}{lcl}
S & \rightarrow & \text{“[”}\ L\ \text{“]”} \\
L & \rightarrow & N \\
 & | & \varepsilon \\
N & \rightarrow & N\ \text{“,”}\ E \\
 & | & E \\
E & \rightarrow & S \\
 & | & \textsc{Number}
\end{array}
$$

gegeben sind, leistet das Gewünschte. Hier steht L für eine ungeklammerte Liste, deren Elemente durch Kommata getrennt sind, N steht für eine ungeklammerte nicht-leere Liste, deren Elemente durch Kommata getrennt sind und E steht für ein Listen-Element, ist also entweder eine in eckigen Klammern eingefasste Liste oder eine Zahl.

(b) Wir definieren eine abstrakte Klasse `Element`, die sowohl Listen als auch Zahlen umfasst:

```
1     Element = MyList(List<Element> listExpr)
2             + MyNumber(Integer number);
```

(c) Die *JavaCup*-Spezifikation des Parsers hat die folgende Form:

```
1  import java_cup.runtime.*;
2  import java.util.*;
3
4  terminal         OPEN, CLOSE, COMMA;
5  terminal Integer NUMBER;
6
7  nonterminal Element       s;
8  nonterminal List<Element> l, n;
9  nonterminal Element       e;
10
11 s ::= OPEN l:e CLOSE
12       {: List<Listen> list = new LinkedList<Listen>();
13          RESULT = new MyList(e);
14       :};
15
16 l ::= n:l               {: RESULT = l; :}
17    |  /* epsilon */     {: List<Element> l = new LinkedList<Element>();
18                           RESULT = l;
19                        :}
20    ;
21
22 n ::= n:l COMMA e:x     {: l.add(x); RESULT = l; :}
23    |  e:x               {: List<Element> l = new LinkedList<Element>();
24                           l.add(x);
25                           RESULT = l;
26                        :}
27    ;
28
29 e ::= s:x               {: RESULT = x;               :}
30    |  NUMBER:x          {: RESULT = new MyNumber(x); :}
31    ;
```

Der dazugehörige Scanner ist wie folgt definiert:

```
1  import java_cup.runtime.*;
2
3  %%
4
5  %char
6  %line
7  %column
8  %cup
9
10 %{
11     private Symbol symbol(int type) {
12         return new Symbol(type, yychar, yychar + yylength());
13     }
14
15     private Symbol symbol(int type, Object value) {
16         return new Symbol(type, yychar, yychar + yylength(), value);
17     }
18 %}
19
20 %%
21
22 "["      { return symbol( sym.OPEN  ); }
23 "]"      { return symbol( sym.CLOSE ); }
24 ","      { return symbol( sym.COMMA ); }
25
26 [0-9]+   { return symbol( sym.NUMBER, new Integer(yytext()) ); }
27
28 [ \t\n]  { /* skip white space */ }
29
30 [^]      { throw new Error("Illegal character"); }
```

Aufgabe 4: Der Typ $\texttt{list}(T)$ sei wie folgt definiert:

```
type list(X) := nil + cons(X, list(X));
```

Die Funktion `addLast` habe die folgende Signatur:

```
signature addLast: list(T) * T -> list(T);
```

und die Variablen `x` und `z` haben den Typ `int`.

(a) Berechnen Sie

$$\mathit{typeEqs}\big(\texttt{addLast(cons(x, nil), z): list(int)}\big).$$

(b) Lösen Sie die in Teil (a) berechneten Typ-Gleichungen.

Lösung:

(a) Wir berechnen zunächst die Typ-Gleichungen nach der im Skript angegebenen Definition.

$$\begin{aligned}
& \mathit{typeEqs}(\texttt{addLast(cons(x, nil), z): list(int)}) \\
= \;& \{\texttt{list}(T) = \texttt{list}(\texttt{int})\} \cup \mathit{typeEqs}(\mathit{cons}(x, \mathit{nil}) : \texttt{list}(T)) \cup \mathit{typeEqs}(z : \texttt{int}) \\
= \;& \{\texttt{list}(T) = \texttt{list}(\texttt{int})\} \cup \{\texttt{list}(S) = \texttt{list}(T)\} \cup \\
& \mathit{typeEqs}(x : S) \cup \mathit{typeEqs}(\texttt{nil} : \texttt{list}(S)) \cup \mathit{typeEqs}(z : \texttt{int}) \\
= \;& \{\texttt{list}(T) = \texttt{list}(\texttt{int}),\ \texttt{list}(S) = \texttt{list}(T),\ \texttt{int} = S,\ \texttt{list}(R) = \texttt{list}(S),\ \texttt{int} = \texttt{int}\}
\end{aligned}$$

(b) Wir lösen die oben berechneten Typ-Gleichungen nach dem im Skript angegebenen Verfahren.

$$\begin{aligned}
& \big\langle \{\texttt{list}(T) = \texttt{list}(\texttt{int}),\ \texttt{list}(S) = \texttt{list}(T),\ \texttt{int} = S,\ \texttt{list}(R) = \texttt{list}(S),\ \texttt{int} = \texttt{int}\},\ [] \big\rangle \\
\rightsquigarrow \;& \big\langle \{T = \texttt{int},\ \texttt{list}(S) = \texttt{list}(T),\ \texttt{int} = S,\ \texttt{list}(R) = \texttt{list}(S),\ \texttt{int} = \texttt{int}\},\ [] \big\rangle \\
\rightsquigarrow \;& \big\langle \{\texttt{list}(S) = \texttt{list}(\texttt{int}),\ \texttt{int} = S,\ \texttt{list}(R) = \texttt{list}(S),\ \texttt{int} = \texttt{int}\},\ [T \mapsto \texttt{int}] \big\rangle \\
\rightsquigarrow \;& \big\langle \{S = \texttt{int},\ \texttt{int} = S,\ \texttt{list}(R) = \texttt{list}(S),\ \texttt{int} = \texttt{int}\},\ [T \mapsto \texttt{int}] \big\rangle \\
\rightsquigarrow \;& \big\langle \{\texttt{int} = \texttt{int},\ \texttt{list}(R) = \texttt{list}(\texttt{int}),\ \texttt{int} = \texttt{int}\},\ [T \mapsto \texttt{int},\ S \mapsto \texttt{int}] \big\rangle \\
\rightsquigarrow \;& \big\langle \{\texttt{list}(R) = \texttt{list}(\texttt{int}),\ \texttt{int} = \texttt{int}\},\ [T \mapsto \texttt{int},\ S \mapsto \texttt{int}] \big\rangle \\
\rightsquigarrow \;& \big\langle \{R = \texttt{int},\ \texttt{int} = \texttt{int}\},\ [T \mapsto \texttt{int},\ S \mapsto \texttt{int}] \big\rangle \\
\rightsquigarrow \;& \big\langle \{\texttt{int} = \texttt{int}\},\ [T \mapsto \texttt{int},\ S \mapsto \texttt{int},\ R \mapsto \texttt{int}] \big\rangle \\
\rightsquigarrow \;& \big\langle \{\},\ [T \mapsto \texttt{int},\ S \mapsto \texttt{int},\ R \mapsto \texttt{int}] \big\rangle
\end{aligned}$$

Damit ist die Substitution $[T \mapsto \texttt{int},\ S \mapsto \texttt{int},\ R \mapsto \texttt{int}]$ eine Lösung der Typ-Gleichungen und wir können folgern, dass der Term tatsächlich den angegebenen Typ hat.

Aufgabe 5: Nehmen Sie an, dass die im Skript eingeführte Sprache *Integer*-`C` um eine `do-while`-Schleife erweitert werden soll, deren Syntax durch die folgende Grammatik-Regel gegeben ist:

statement → "`do`" *statement* "`while`" "`(`" *boolExpr* "`)`" .

Die Semantik dieses Konstruktes soll mit der Semantik des entsprechenden Konstruktes in der Sprache `C` übereinstimmen.

(a) Geben Sie eine Gleichung an, die beschreibt, wie eine `do-while`-Schleife in *Java-Byte-Code* übersetzt werden kann.

(b) Geben Sie die Methode *compile*() an, die das entsprechende Konstrukt übersetzt. Gehen Sie dabei davon aus, dass Sie diese Methode innerhalb einer Klasse `DoWhile` implementieren, wobei diese Klasse für Ep wie folgt spezifiziert ist:

`Statement` = ··· + `DoWhile`(`Statement` *stmnt*, `BoolExpr` *cond*) + ···;

Lösung:

(a) Die Übersetzung einer `do-while`-Schleife der Form

`while` (*cond*) *statement*

orientiert sich an der folgenden Spezifikation:

$$
\begin{array}{lcl}
\mathit{compile}(\texttt{do}\ \mathit{stmnt}\ \texttt{while}\ (\mathit{cond})\ ) & = & [\ \mathit{loop}\texttt{:}\ ] \\
& + & \mathit{stmnt}.\mathit{compile}() \\
& + & \mathit{cond}.\mathit{compile}() \\
& + & [\ \texttt{ifeq}\ \mathit{next}\ ] \\
& + & [\ \texttt{goto}\ \mathit{loop}\ ] \\
& + & [\ \mathit{next}\texttt{:}\ ]
\end{array}
$$

(b) Die Methode *compile*() kann in der Klasse `DoWhile` wie folgt implementiert werden:

```
1     public List<AssemblerCmd> compile() {
2         List<AssemblerCmd> result = new LinkedList<AssemblerCmd>();
3         LABEL        loopLabel = new LABEL();
4         LABEL        nextLabel = new LABEL();
5         AssemblerCmd ifeq      = new IFEQ(nextLabel.getLabel());
6         AssemblerCmd gotoLoop  = new GOTO(loopLabel.getLabel());
7         result.add(loopLabel);
8         result.addAll(mStmnt.compile());
9         result.addAll(mCond.compile());
10        result.add(ifeq);
11        result.add(gotoLoop);
12        result.add(nextLabel);
13        return result;
14      }
```

Aufgabe 6: Zeigen Sie, dass die wie folgt definierte Menge M

$$M := \{x \in \mathbb{R} \mid 0 \leq x \leq 1\}$$

nicht abzählbar ist.
Hinweis: Nehmen Sie an, dass Sie eine Funktion

$$f : \mathbb{N} \to M$$

gibt, so dass es für jedes $x \in M$ eine Zahl $n \in \mathbb{N}$ gibt, für die $x = f(n)$ ist. Führen Sie diese Annahme zum Widerspruch, indem Sie eine Zahl $\gamma \in M$ konstruieren, für die

$$f(n) \neq \gamma \quad \text{für alle } n \in \mathbb{N}$$

gilt. Zur Konstruktion von γ ist es sinnvoll die Hilfsfunktion

$$\mathit{digit} : M \times \mathbb{N} \to \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$$

zu benutzen, die so definiert ist, dass $\mathit{digit}(x, k)$ die k-te Ziffer der Zahl x ist. Beispielsweise gilt

$$\mathit{digit}(0.125, 1) = 1, \quad \mathit{digit}(0.125, 2) = 2, \quad \mathit{digit}(0.125, 3) = 5, \quad \text{und}$$
$$\mathit{digit}(0.125, k) = 0, \quad \text{für alle } k > 3.$$

Lösung: Wir definieren die Zahl γ indem wir die einzelnen Ziffern von γ durch

$$\mathit{digit}(\gamma, n) = \big(\mathit{digit}\big(f(n), n\big) + 1\big) \texttt{ mod } 10$$

definieren. Den Widerspruch erhalten wir nun aus der Tasache, dass es eine Zahl $n_0 \in \mathbb{N}$ geben muss, für die

$$f(n_0) = \gamma$$

gilt. Betrachten wir nun die n_0-te Ziffer von γ, so sehen wir

$$\begin{aligned} \mathit{digit}(f(n_0), n_0) &= \mathit{digit}(\gamma, n_0) \\ &= (\mathit{digit}(f(n_0), n_0) + 1) \texttt{ mod } 10. \end{aligned}$$

Da für jede Ziffer $x \neq (x + 1) \texttt{ mod } 10$ gilt, ist dies ein Widerspruch, der uns zeigt, dass die Menge M nicht abzählbar ist.

Aufgabe 7: Aus wieviel Zeichen muss der kleinste Ausdruck e mindestens bestehen, für den

$$\mathit{ershov}(e) = 30$$

gilt?

Lösung: Wir bezeichnen die Anzahl der Zeichen, die der kleineste Ausdruck mit Ershov-Zahl n haben muss, mit a_n. Dann haben wir $a_1 = 1$ und

$$a_{n+1} = 2 * a_n + 1$$

Diese Rekurrenz-Gleichung hat die Lösung $a_n = 2^n - 1$, was wir leicht durch Induktion nachweisen könnten. Setzen wir für n den Wert 30 ein, so sehen wir, dass der kleinste Ausdruck aus

$$2^{30} - 1 = 1\,073\,741\,823$$

Zeichen besteht. Dieser Ausdruck hätte daher eine Größe von knapp einem Gigabyte und wir können schließen, dass in der Praxis 30 Register für die Auswertung von Ausdrücken ausreichen.