Aufgaben zur Klausurvorbereitung für die Vorlesung "*Compilerbau*"

Hinweis: Bei der Klausur müssen alle Aufgabenbläter mit abgegeben werden, sonst ist die Klausur ungültig!

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

$$E \rightarrow E\ E\ \text{“+”} \mid E\ E\ \text{“*”} \mid \text{“x”} .$$

Wenden Sie den Algorithmus von Earley auf den String "xx+x*" an.

Aufgabe 2: Die Grammatik $G = \langle \{S\}, \{+, -, \mathtt{a}\}, R, S \rangle$ habe die folgenden Regeln:

$$S \rightarrow S\ S + \mid S\ S - \mid \mathtt{a}.$$

(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.

(e) Untersuchen Sie, ob diese Grammatik mehrdeutig ist.

Aufgabe 3: Die Grammatik $G = \langle \{A, B\}, \{ \text{“u”} , \text{“x”} , \text{“y”} , \text{“z”} \}, R, A \rangle$ habe die folgenden Regeln:

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

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.

Aufgabe 4: 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) Definieren Sie geeignete Klassen, mit deren Hilfe Sie geschachtelte Listen repräsentieren können.

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

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

Aufgabe 5: Der Typ `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.

Aufgabe 6: 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:

$$\mathit{statement} \rightarrow \text{“}\texttt{do}\text{”}\ \mathit{statement}\ \text{“}\texttt{while}\text{”}\ \ \text{“}\texttt{(}\text{”}\ \mathit{boolExpr}\ \text{“}\texttt{)}\text{”}\ .$$

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:

$$\texttt{Statement} \ = \ \cdots \ + \ \texttt{DoWhile(Statement}\ \mathit{stmnt}, \ \texttt{BoolExpr}\ \mathit{cond}) \ + \ \cdots;$$