Lösungen zu den Aufgaben zur Klausurvorbereitung

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

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

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

Lösung: Wir definieren $s :=$ “xx+x*”. Der Earley-Algorithmus berechnet die folgenden Mengen:

(a) $Q_0 = \{\ \langle \widehat{S} \to \bullet\ E, 0\rangle,$
$\langle E \to \bullet\ E\ E\ \text{“+”}, 0\rangle,$
$\langle E \to \bullet\ E\ E\ \text{“*”}, 0\rangle,$
$\langle E \to \bullet\text{“x”}, 0\rangle\ \}$

(b) Wegen $s[1] =$ “x” gilt:
$Q_1 = \{\ \langle E \to \text{“x”}\bullet, 0\rangle,$
$\langle \widehat{S} \to E\ \bullet, 0\rangle,$
$\langle E \to E \bullet E\ \text{“+”}, 0\rangle,$
$\langle E \to E \bullet E\ \text{“*”}, 0\rangle,$
$\langle E \to \bullet\ E\ E\ \text{“+”}, 1\rangle,$
$\langle E \to \bullet\ E\ E\ \text{“*”}, 1\rangle,$
$\langle E \to \bullet\text{“x”}, 1\rangle\ \}$

(c) Wegen $s[2] =$ “x” gilt:
$Q_2 = \{\ \langle E \to \text{“x”}\bullet, 1\rangle,$
$\langle E \to E\ E \bullet\ \text{“*”}, 0\rangle,$
$\langle E \to E\ E \bullet\ \text{“+”}, 0\rangle,$
$\langle E \to E \bullet E\ \text{“*”}, 1\rangle,$
$\langle E \to E \bullet E\ \text{“+”}, 1\rangle,$
$\langle E \to \bullet\ E\ E\ \text{“*”}, 2\rangle,$
$\langle E \to \bullet\ E\ E\ \text{“+”}, 2\rangle,$
$\langle E \to \bullet\text{“x”}, 2\rangle\ \}$

(d) Wegen $s[3] =$ “+” gilt:
$Q_3 = \{\ \langle E \to E\ E\ \text{“+”}\bullet, 0\rangle,$
$\langle \widehat{S} \to E\ \bullet, 0\rangle,$
$\langle E \to E \bullet E\ \text{“*”}, 0\rangle,$
$\langle E \to E \bullet E\ \text{“+”}, 0\rangle,$
$\langle E \to \bullet\ E\ E\ \text{“*”}, 3\rangle,$
$\langle E \to \bullet\ E\ E\ \text{“+”}, 3\rangle,$
$\langle E \to \bullet\text{“x”}, 3\rangle\ \}$

(e) Wegen $s[4] =$ “x” gilt:
$Q_4 = \{\ \langle E \to \text{“x”}\bullet, 3\rangle,$
$\langle E \to E\ E \bullet\ \text{“*”}, 0\rangle,$
$\langle E \to E\ E \bullet\ \text{“+”}, 0\rangle$
$\langle E \to E \bullet E\ \text{“*”}, 3\rangle,$
$\langle E \to E \bullet E\ \text{“+”}, 3\rangle,$
$\langle E \to \bullet\ E\ E\ \text{“*”}, 4\rangle,$
$\langle E \to \bullet\ E\ E\ \text{“+”}, 4\rangle,$
$\langle E \to \bullet\text{“x”}, 4\rangle\ \}$

(f) Wegen $s[5] =$ "$*$" gilt:

$$\begin{aligned} Q_5 = \{\ & \langle E \to E\ E\ \text{“}*\text{”}\bullet, 0\rangle, \\ & \langle \widehat{S} \to E\ \bullet, 0\rangle, \\ & \langle E \to E \bullet E\ \text{“}*\text{”}, 0\rangle, \\ & \langle E \to E \bullet E\ \text{“}+\text{”}, 0\rangle, \\ & \langle E \to \bullet\ E\ E\ \text{“}*\text{”}, 5\rangle, \\ & \langle E \to \bullet\ E\ E\ \text{“}+\text{”}, 5\rangle, \\ & \langle E \to \bullet\text{“}\texttt{x}\text{”}, 5\rangle\ \} \end{aligned}$$

Da dieser Zustand das Earley-Item $\langle \widehat{S} \to E\ \bullet, 0\rangle$ enthält, liegt der String s in der von der Grammatik G erzeugten Sprache.

Aufgabe 2: Die Grammatik $G = \langle \{E\}, \{\text{“}+\text{”}, \text{“}-\text{”}, \text{“}\texttt{a}\text{”}\}, R, E\rangle$ habe die folgenden Regeln:

$E \to E\ E$ “+” | $E\ E$ “-” | “a”.

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

(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.

Lösung:

(a) Es gilt offenbar

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

Bemerkung: Das Zeichen “$” liegt in der Menge $\mathit{Follow}(E)$, weil E das Start-Symbol der Grammatik ist.

(b) Wir erhalten die folgenden Zustände:

1. Wir definieren $s_0 = \mathit{closure}\big(\{\widehat{S} \to \star E\}\big)$ und finden

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

2. Wir definieren $s_1 = \mathit{goto}(s_0, E)$ und finden

$$\begin{aligned} s_1 = \{\ & \widehat{S} \to E\ \bullet, \\ & E \to E \bullet E\text{“}+\text{”}, \\ & E \to E \bullet E\text{“}-\text{”}, \\ & E \to \bullet\ E\ E\text{“}+\text{”}, \\ & E \to \bullet\ E\ E\text{“}-\text{”}, \\ & E \to \bullet\text{“}\texttt{a}\text{”} \\ \}. & \end{aligned}$$

3. Wir definieren $s_2 = \mathit{goto}(s_1, E)$ und finden

$$\begin{aligned} s_2 = \{\ & E \to E\ E \bullet \text{“}+\text{”}, \\ & E \to E\ E \bullet \text{“}-\text{”}, \\ & E \to E \bullet E\text{“}+\text{”}, \\ & E \to E \bullet E\text{“}-\text{”}, \\ & E \to \bullet\ E\ E\text{“}+\text{”}, \\ & E \to \bullet\ E\ E\text{“}-\text{”}, \\ & E \to \bullet\text{“}\texttt{a}\text{”} \\ \}. & \end{aligned}$$

4. Wir definieren $s_3 = \mathit{goto}(s_2, \text{“}\texttt{a}\text{”})$ und finden
$$s_3 = \{E \to \text{“}\texttt{a}\text{”}\bullet\}.$$
5. Wir definieren $s_4 = \mathit{goto}(s_2, \text{“}\texttt{+}\text{”})$ und finden
$$s_4 = \{E \to E\, E\text{“}\texttt{+}\text{”}\bullet\}.$$
6. Wir definieren $s_5 = \mathit{goto}(s_2, \text{“}\texttt{-}\text{”})$ und finden
$$s_5 = \{E \to E\, E\text{“}\texttt{-}\text{”}\bullet\}.$$
7. Als nächstes berechnen wir:
   i. $\mathit{goto}(s_0, \text{“}\texttt{a}\text{”}) = s_3$.
   ii. $\mathit{goto}(s_1, \text{“}\texttt{a}\text{”}) = s_3$.
   iii. $\mathit{goto}(s_2, E) = s_2$.

   Damit haben wir nun alle interessanten Werte der Funktion *goto*() berechnet, denn für alle bisher nicht explizit angegebenen Werte liefert diese Funktion die leere Menge.

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

1. $\mathit{action}(s_0, \text{“}\texttt{a}\text{”}) = \langle \texttt{shift}, s_3 \rangle$
2. $\mathit{action}(s_1, \text{“}\texttt{\$}\text{”}) = \texttt{accept}$
3. $\mathit{action}(s_1, \text{“}\texttt{a}\text{”}) = \langle \texttt{shift}, s_3 \rangle$
4. $\mathit{action}(s_2, \text{“}\texttt{+}\text{”}) = \langle \texttt{shift}, s_4 \rangle$
5. $\mathit{action}(s_2, \text{“}\texttt{-}\text{”}) = \langle \texttt{shift}, s_5 \rangle$
6. $\mathit{action}(s_2, \text{“}\texttt{a}\text{”}) = \langle \texttt{shift}, s_3 \rangle$
7. $\mathit{action}(s_3, \text{“}\texttt{\$}\text{”}) = \langle \texttt{reduce}, E \to \text{“}\texttt{a}\text{”} \rangle$
8. $\mathit{action}(s_3, \text{“}\texttt{+}\text{”}) = \langle \texttt{reduce}, E \to \text{“}\texttt{a}\text{”} \rangle$
9. $\mathit{action}(s_3, \text{“}\texttt{-}\text{”}) = \langle \texttt{reduce}, E \to \text{“}\texttt{a}\text{”} \rangle$
10. $\mathit{action}(s_3, \text{“}\texttt{a}\text{”}) = \langle \texttt{reduce}, E \to \text{“}\texttt{a}\text{”} \rangle$
11. $\mathit{action}(s_4, \text{“}\texttt{\$}\text{”}) = \langle \texttt{reduce}, E \to E\, E\text{“}\texttt{+}\text{”} \rangle$
12. $\mathit{action}(s_4, \text{“}\texttt{+}\text{”}) = \langle \texttt{reduce}, E \to E\, E\text{“}\texttt{+}\text{”} \rangle$
13. $\mathit{action}(s_4, \text{“}\texttt{-}\text{”}) = \langle \texttt{reduce}, E \to E\, E\text{“}\texttt{+}\text{”} \rangle$
14. $\mathit{action}(s_4, \text{“}\texttt{a}\text{”}) = \langle \texttt{reduce}, E \to E\, E\text{“}\texttt{+}\text{”} \rangle$
15. $\mathit{action}(s_5, \text{“}\texttt{\$}\text{”}) = \langle \texttt{reduce}, E \to E\, E\text{“}\texttt{-}\text{”} \rangle$
16. $\mathit{action}(s_5, \text{“}\texttt{+}\text{”}) = \langle \texttt{reduce}, E \to E\, E\text{“}\texttt{-}\text{”} \rangle$
17. $\mathit{action}(s_5, \text{“}\texttt{-}\text{”}) = \langle \texttt{reduce}, E \to E\, E\text{“}\texttt{-}\text{”} \rangle$
18. $\mathit{action}(s_5, \text{“}\texttt{a}\text{”}) = \langle \texttt{reduce}, E \to E\, E\text{“}\texttt{-}\text{”} \rangle$

Für die Funktion *goto*() finden wir:

1. $\mathit{goto}(s_0, E) = s_1$
2. $\mathit{goto}(s_1, E) = s_2$
3. $\mathit{goto}(s_2, E) = s_2$

(d) Wir erhalten die folgenden Zustände:

1. Wir setzen wieder $s_0 = \mathit{closure}\Big(\{\widehat{S} \to \bullet E : \$\}\Big)$ und erhalten diesmal
$$\begin{array}{lll} s_0 = & \{ & \widehat{S} \to \bullet\, E : \text{“}\texttt{\$}\text{”}, \\ & & E \to \bullet\, E\; E\text{“}\texttt{+}\text{”} : \{\text{“}\texttt{\$}\text{”}, \text{“}\texttt{a}\text{”}\}, \\ & & E \to \bullet\, E\; E\text{“}\texttt{-}\text{”} : \{\text{“}\texttt{\$}\text{”}, \text{“}\texttt{a}\text{”}\}, \\ & & E \to \bullet\text{“}\texttt{a}\text{”} : \{\text{“}\texttt{\$}\text{”}, \text{“}\texttt{a}\text{”}\} \\ & \}. & \end{array}$$

2. Wir definieren $s_1 = goto(s_0, E)$ und erhalten

$$\begin{aligned} s_1 \;=\; \{\quad & \widehat{S} \to E\bullet : \text{“\$”}, \\ & E \to E \bullet E\text{“+”} : \{\text{“\$”}, \text{“a”}\}, \\ & E \to E \bullet E\text{“-”} : \{\text{“\$”}, \text{“a”}\}, \\ & E \to \bullet\, E\ E\text{“+”} : \{\text{“+”}, \text{“-”}, \text{“a”}\}, \\ & E \to \bullet\, E\ E\text{“-”} : \{\text{“+”}, \text{“-”}, \text{“a”}\}, \\ & E \to \bullet\text{“a”} : \{\text{“+”}, \text{“-”}, \text{“a”}\} \\ \}. & \end{aligned}$$

3. Wir definieren $s_2 = goto(s_1, E)$ und erhalten

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

4. Wir definieren $s_3 = goto(s_2, E)$ und erhalten

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

   Beachten Sie, dass $s_2 \neq s_3$ ist, denn die Menge der Folge-Token sind für die markierten Regeln $E \to E\ E \bullet$ “+” und $E \to E\ E \bullet$ “-” in den Mengen s_2 und s_3 unterschiedlich.

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

$$s_4 = \{E \to \text{“a”}\bullet : \{\text{“\$”}, \text{“a”}\}\}.$$

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

$$s_5 = \{E \to \text{“a”}\bullet : \{\text{“+”}, \text{“-”}, \text{“a”}\}\}$$

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

$$s_6 = \{E \to E\,E\text{“+”}\bullet : \{\text{“\$”}, \text{“a”}\}\}$$

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

$$s_7 = \{E \to E\,E\text{“+”}\bullet : \{\text{“+”}, \text{“-”}, \text{“a”}\}\}$$

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

$$s_8 = \{E \to E\,E\text{“-”}\bullet : \{\text{“\$”}, \text{“a”}\}\}$$

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

$$s_9 = \{E \to E\,E\text{“-”}\bullet : \{\text{“+”}, \text{“-”}, \text{“a”}\}\}$$

(e) Bei der Berechnung der Funktion *action*() in Teil (c) der Aufgabe haben wir gesehen, dass es keine Konflikte gibt. Daher ist die Grammatik eine SLR-Grammatik und damit sicher nicht mehrdeutig.

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

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

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, ersetzen wir zunächst die Variable B durch ihre Definition. Für A erhalten wir dann die folgenden Grammatik-Regeln:

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

Basteln wir daraus nun, wie im Skript beschrieben, einen endlichen Automaten, so ist leicht zu sehen, dass die akzeptierenden Zustände homogen sind. Damit handelt es sich bei der angegebenen Grammatik um 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

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

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

$$s_1 = \{A \to \text{“u”} \bullet \text{“z”},\ B \to \text{“u”} \bullet\}.$$

Bei der Berechnung von $\mathit{action}(s_1, \text{“z”})$ tritt nun eine Shift-Reduce-Konflikt auf, denn es gilt

$$\mathit{follow}(B) = \{\text{“x”}, \text{“z”}\}.$$

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 4: 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.

Lösung:

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

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

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

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

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

do statement `while` (*cond*) "`;`"

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  }
```