Aufgabe 1 (Reguläre Ausdrücke und endliche Automaten) *12 Punkte*

Betrachten Sie die folgenden Sprachen:

(i) $L_1 = \{w \in \{\mathbf{a}, \mathbf{b}, \mathbf{c}\}^* :$ in w folgt niemals ein **c** direkt auf **a**$\}$

(ii) $L_2 = \{w \in \{\mathbf{a}, \mathbf{b}, \mathbf{c}\}^* :$ die Anzahl an **a**'s in w ist durch 3 teilbar$\}$

(iii) $L_3 = \{w \in \{\mathbf{a}, \mathbf{b}, \mathbf{c}\}^* :$ die Länge jeder maximalen Folge aufeinanderfolgender **a**'s in w ist gerade$\}$

(iv) $L_4 = \{w \in \{\mathbf{a}, \mathbf{b}, \mathbf{c}\}^* :$ in w steht kein **a** vor einem **c** (auch nicht indirekt)$\}$

(v) $L_5 = \{w \in \{\mathbf{a}, \mathbf{b}, \mathbf{c}\}^* :$ jeder Folge **a**'s folgt direkt mindestens ein **c**$\}$

(a) Geben Sie für die oben angegebenen Sprachen reguläre Ausdrücke an, die diese Sprachen definieren. *6 Punkte*

(b) Geben Sie für die obigen Sprachen deterministische endliche Automaten an, die diese Sprachen erkennen. Eine Angabe des Zustandsdiagramms ist hier ausreichend. *6 Punkte*

Aufgabe 2 (Top-Down-Parser) *12 Punkte*

Gegeben sei die Grammatik $G = (\{A, L, B, C, R, S\}, \{\mathbf{s}, \mathbf{w}, \mathbf{i}, \mathbf{f}, \mathbf{t}, \mathbf{c}, \mathbf{a}, \mathbf{o}\}, P, S)$ mit

$$
\begin{aligned}
P = \{ \quad & S \to \mathbf{s}A\mathbf{f}R\mathbf{w}B \\
& A \to \mathbf{i}L, \\
& B \to \mathbf{ici}C, \\
& C \to \mathbf{a}B \mid \mathbf{o}B \mid \varepsilon, \\
& L \to \mathbf{ti}L \mid \varepsilon, \\
& R \to \mathbf{i}L\}
\end{aligned}
$$

(a) Berechnen Sie die FIRST_1-Mengen aller Nichtterminale und geben Sie die initialen Steuermengen aller Produktionen an. *4 Punkte*

(b) Bestimmen Sie die FOLLOW-Mengen aller Nichtterminale entsprechend des im Kurs angegebenen Algorithmus. Geben Sie hierzu den Graphen sowie die FOLLOW-Mengen der Nichtterminale an. *5 Punkte*

(c) Geben Sie die Parse-Tabelle für diese Grammatik an. *3 Punkte*

Aufgabe 3 (Bottom-Up-Parser) *18 Punkte*

Gegeben sei die Grammatik $G = (\{A, B, C, D, E\}, \{\mathbf{s}, \mathbf{t}, \mathbf{u}, \mathbf{v}, \mathbf{w}, \mathbf{x}, \mathbf{y}, \mathbf{z}\}, P, A)$ mit

$$
\begin{aligned}
P = \{ \quad & A \rightarrow B, \\
& B \rightarrow \mathbf{s}C\mathbf{t}C\mathbf{u}D, \\
& C \rightarrow C\mathbf{w}\mathbf{v} \mid \mathbf{v}, \\
& D \rightarrow D\mathbf{x}E \mid D\mathbf{y}E \mid E, \\
& E \rightarrow \mathbf{z} \}
\end{aligned}
$$

Erzeugen Sie einen $LR(1)$-Parser für G. Geben Sie dazu die kanonische $LR(1)$-Kollektion sowie die Steuertabelle an.

Aufgabe 4 (Syntaxanalyse) *10 Punkte*

`SELECT` *list* `FROM` *list* `WHERE` *boolexpr*

ist die stark vereinfachte Grundform einer SQL-Anfrage, wobei *list* jeweils aus einem oder mehreren durch Kommas voneinander getrennten Objektbezeichnern besteht und *boolexpr* entweder aus einem einfachen Vergleich `ID COMP ID` oder einer Verkettung von einfachen Vergleichen mit `AND` oder `OR` besteht. Beispiele für zulässige Ausdrücke sind:

- `SELECT ID, ID, ID FROM ID WHERE ID COMP ID`
- `SELECT ID FROM ID, ID, ID WHERE ID COMP ID AND ID COMP ID OR ID COMP ID`

Es ist noch nicht entschieden, ob die Syntaxanalyse mit einem Top-Down-Parser oder einem Bottom-Up-Parser erfolgen soll. Um für beide Fälle gerüstet zu sein, sollen Sie deshalb sowohl eine LL(1)-Grammatik als auch eine LR(1)-Grammatik angeben, die die Syntax von SQL-Anfragen analysieren. Ein formaler Beweis der LL(1)- bzw. LR(1)-Eigenschaft ist nicht erforderlich. *je 5 Punkte*

Aufgabe 5 (Syntaxgesteuerte Definition) *16 Punkte*

In dieser Aufgabe soll eine Grammatik entworfen werden, die in der Lage ist, eine gegebene Zuweisung zu berechnen. Dazu wird einer Variablen sowohl ein Wert als auch der passende Typ zugewiesen. Mögliche Typen sind hierbei *real*, *bool* und *error*. Eine Zuweisung hat dabei die Form:

id := *expr*;

Vom Scanner wird folgende Tokenmenge erkannt: {**id**, **:=**, **num**, **;**, **+** , **-**, **<**, **=** , **and**, **or**, **not** }. **num** stellt hierbei eine Besonderheit dar, da diesem Token bereits durch

den Scanner ein Attribut *val* zugeordnet wird. Die restlichen Symbole sind mit der üblichen Semantik versehen. Vergleiche sind nur auf numerischen Ausdrücken erlaubt. Die Ausdrücke selbst sind in Postfix-Notation gegeben.

(a) Geben Sie eine Grammatik an, mit Hilfe derer ein gegebener Ausdruck gemäß oben stehender Beschreibung erkannt wird. (Sie können diese Aufgabe auch zusammen mit Aufgabenteil (b) lösen.) *4 Punkte*

(b) Geben Sie semantische Regeln an, so dass das Ergebnis in den Attributen des Tokens **id** (*realval*, *boolval*, *type*) gespeichert wird. Die Regeln können in algorithmischer Form angegeben werden, z.B. **if then else** Konstrukte verwenden. *10 Punkte*

(c) Welche Attribute der Grammatik sind synthetisiert, welche vererbt? *2 Punkte*

Aufgabe 6 (Erzeugen von 3-Adress-Code) *8 Punkte*

Gegeben sei folgendes Programm der im Kurstext eingeführten, imperativen Beispielsprache:

```
program main {

 function fib(n : integer) : integer {
   var integer n1;
   var integer n2;
   var integer f1;
   var integer f2;
   if n = 0
     return 0
   end;
   if n = 1
     return 1
   end;
   n1 := n - 1;
   n2 := n - 2;
   f1 := fib(n1);
   f2 := fib(n2);
   return f1 + f2;
 }

 var integer x;
 const integer y:= 7;
 x := fib(y);
}
```

Übersetzen Sie das Hauptprogramm und die Funktion *fib* direkt in äquivalenten 3-Adress-Code. Verwenden Sie die im Kurs eingeführte Grundvariante des 3-Adress-Code mit den Erweiterungen der 8. Gruppe zur Behandlung von Prozedur- und

Funktionsaufrufen. Wir setzen voraus, dass die Feldgröße des Typs *integer* der Register- und Speicherwortgröße des Prozessors entspricht.

Aufgabe 7 (Datenflussanalyse)

24 Punkte

Folgender 3-Address-Code beschreibt den Stein'schen Algorithmus zur ggT-Berechnung (die Funktion erhält zwei Parameter in T1 und T2 und berechnet ggT(T1,T2). Das Ergebnis steht am Ende in Register T2):

Block	Begründung für Blockanfang	Zeile	3AC-Anweisung
		(1)	`T3 := 0`
		(2)	`T5 := T2 mod 2`
		(3)	`T6 := T1 mod 2`
		(4)	`if T5 = 1 goto 12`
		(5)	`if T6 = 1 goto 10`
		(6)	`T1 := T1 / 2`
		(7)	`T2 := T2 / 2`
		(8)	`T3 := T3 + 1`
		(9)	`goto 2`
		(10)	`T4 := -T2`
		(11)	`goto 13`
		(12)	`T4 := T1`
		(13)	`if T4 = 0 goto 24`
		(14)	`T6 := T4 mod 2`
		(15)	`if T6 = 1 goto 18`
		(16)	`T4 := T4 / 2`
		(17)	`goto 14`
		(18)	`if T4 > 0 goto 21`
		(19)	`T2 := -T4`
		(20)	`goto 22`
		(21)	`T1 := T4`
		(22)	`T4 := T1 - T2`
		(23)	`goto 13`
		(24)	`if T3 = 0 goto 30`
		(25)	`T2 := T1`
		(26)	`if T3 < 2 goto 31`
		(27)	`T2 := T2 * T1`
		(28)	`T3 := T3 - 1`
		(29)	`goto 26`
		(30)	`T2 := 1`
		(31)	`return T2`

(a) Markieren Sie die Basisblöcke des 3AC-Programms und nummerieren Sie diese von oben nach unten, beginnend mit 1, durch. Begründen Sie, warum gerade an diesen Stellen ein Basisblock beginnt. *5 Punkte*

Hinweis: Abweichend von den „Hinweisen zur Bearbeitung der Klausur“ dürfen Sie die Blöcke direkt im obenstehenden Aufgabentext markieren und die Blocknummern und Begründungen annotieren. *Vergessen Sie in diesem Falle nicht, dieses Aufgabenblatt mit Matrikelnummer und Namen zu versehen und mit abzugeben.*

(b) Geben Sie den Flussgraphen für das 3AC-Programm an. *3 Punkte*

(c) Nach welcher Systematik kann man Datenflussanalysen kategorisieren? Führen Sie vier Arten systematisch auf und nennen Sie jeweils kurz ein Anwendungsbeispiel. *3 Punkte*

(d) Zur späteren Optimierung benötigen wir die Menge aller vielbeschäftigten Ausdrücke (*very busy expressions*) für die Basisblöcke des 3AC-Programms. Zu deren Ermittlung führen wir eine Datenflussanalyse durch. Geben Sie dazu die *gen*- und *kill*-Mengen für sämtliche Basisblöcke des Programms an. Stellen Sie dann die Datenflussgleichungen (*in*- und *out*-Mengen) für die gegebene Aufgabe auf. Lösen Sie schließlich das Datenflussgleichungssystem iterativ. Wie sind hier die initialen *in*-Mengen zu wählen? *13 Punkte*