6 EBNF

Punkte	
(a)	/ 4
(b)	/11
$\sum$	/15

Hinweis: Sie dürfen die Zirkumflexe ˆ in EBNF-Termen weglassen.

(a) Sei $\Sigma = \{a, b, c, d\}$. Geben Sie die Mengen V und R einer EBNF-Definition $\mathcal{E} = (V, \Sigma, S, R)$ an, so dass

$$W(\mathcal{E}) = \{\ (ab)^n c^k d^{2m} \mid m, n \geq 0, k \geq m + n\ \}.$$

$V = \{\ S,$

$R = \{$

(b) Sei $\mathcal{E} = (V, \{a, b\}, S, R)$ eine EBNF-Definition mit $V = \{S\}$ und $R = \{\ S ::= \hat{[}\, a\, \hat{(}Sb\hat{|}\, Sbb\hat{)}\hat{]}\ \}$. Dokumentieren Sie die ersten drei Iterationsschritte der Fixpunktsemantik von $\mathcal{E}$.

$\Big(\emptyset\Big) \mapsto$

Sei $\rho\colon V \to \mathcal{P}(\Sigma^*)$ mit $\rho(S) = \{a^n b^m \mid 2n \geq m \geq n \geq 0\}$. Zeigen Sie, dass die Gleichung $[\![\, \hat{[}\, a\, \hat{(}Sb\hat{|}Sbb\hat{)}\hat{]}\,]\!](\rho) = \rho(S)$ gilt. Wenden Sie dazu zuerst schrittweise die Semantik von EBNF-Termen an und fassen Sie dann geeignet zusammen.

2 Pulsierender Speicher

Punkte	
(a)	/ 5
(b)	/12
$\sum$	/17

Gegeben sei das folgende C-Programm:

```
1  #include <stdio.h>
2  int a;
3
4  void g(int x, int *y);
5
6  void f(int *i, int j) {
7    /*label1*/
8    if (*i + j < a) {
9      *i = *i + 1;
10     f(i, j);            /* $1 */
11   }
12   /*label2*/
13 }
14
15 void g(int x, int *y) {
16   int i = 2;
17   /*label3*/
18   while (x != 1) {
19     f(&i, x);           /* $2 */
20     x = x / 2;
21     *y = *y + 1;
22     /*label4*/
23   }
24 }
25
26 int main() {
27   int x = 0;
28   scanf("%i", &a);
29   /*label5*/
30   g(a, &x);             /* $3 */
31   /*label6*/
32   return 0;
33 }
```

Objektname	Gültigkeitsbereich

(a) Tragen Sie den Gültigkeitsbereich jedes Objektes in die Tabelle ein. Nutzen Sie dazu die Zeilennummern.

(b) Führen Sie jedes der drei folgenden Speicherbelegungsprotokolle um jeweils vier Schritte weiter. Dokumentieren Sie die aktuelle Situation beim Passieren der Marken *label1* bis *label6*. Geben Sie jeweils den Rücksprungmarkenkeller und die **sichtbaren** Variablen mit ihrer Wertebelegung an. Die Inhalte von Speicherzellen nicht sichtbarer Variablen müssen Sie nur bei Änderungen eintragen. Die bereits festgelegten Rücksprungmarken sind *$1* bis *$3*.

HP	RM	Umgebung						
		1	2	3	4	5	6	7
label5	—	a 7	x 0					

HP	RM	Umgebung										
		1	2	3	4	5	6	7	8	9	10	11
label1	2:3	a 8	2	2	2	4	i 5	j 2				

HP	RM	Umgebung										
		1	2	3	4	5	6	7	8	9	10	11
label2	1:1:2:3	a 9	2	2	2	7	5	2	5	2	i 5	j 2

1 Programmieren in C

Punkte	
(a)	/ 8
(b)	/ 5
(c)	/ 7
$\sum$	/20

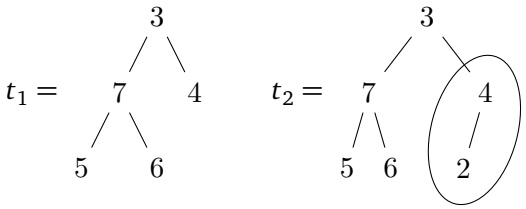
(a) Schreiben Sie ein C-Programm, welches wiederholt zur Eingabe von ganzen Zahlen auffordert bis 0 eingegeben wird. Danach soll das Programm jeweils ausgeben, wie viele der eingegebenen Zahlen (abzüglich der 0) durch 3 teilbar, durch 5 teilbar, sowie durch 3 und durch 5 teilbar waren.

Für die Aufgaben **(b)** und **(c)** gilt folgende Typdefinition für Binärbäume:

```
typedef struct node *tree;
typedef struct node {
  int value;
  tree left, right;
};
```

(b) Schreiben Sie eine Funktion `int contains(tree t, int i)` welche `1` zurück gibt falls der Baum `t` den Wert `i` enthält und sonst `0`.

(c) Ein Baum `t` heißt *projektiv* falls für jeden Teilbaum `s` von `t` gilt: die Menge der Knotenbeschriftungen von `s` ist von der Form $\{i, i+1, \ldots, j\}$, wobei i und j jeweils die minimale bzw. maximale in `s` vorkommende Knotenbeschriftung ist. Zum Beispiel ist der Baum t_1 projektiv, aber t_2 ist nicht projektiv, da der eingekreiste Teilbaum die Bedingung verletzt (in der Menge $\{2,4\}$ der Knotenbeschriftungen fehlt die 3).



Schreiben Sie eine Funktion `int projective(tree t)` die `1` zurück gibt falls `t` projektiv ist und sonst `0`. Sie dürfen folgende Hilfsfunktionen benutzen: `int contains(tree t, int i)` aus Teilaufgabe **(b)** sowie `int maximum(tree t)` und `int minimum(tree t)` welche die maximale bzw. minimale Knotenbeschriftung eines **nicht leeren** Binärbaums ermitteln. Falls Sie eigene Hilfsfunktionen verwenden, geben Sie diese vollständig an!