UNIVERSITÄT KOBLENZ · LANDAU

Theoretische Informatik II

WS 2007/2008

Jun.-Prof. Dr. Bernhard Beckert
Ulrich Koch

Nachklausur

25. 03. 2008

Persönliche Daten — bitte gut leserlich ausfüllen!

Vorname: .....

Nachname: .....

Matrikelnummer: .....

Klausurergebnis — bitte nicht ausfüllen!

Aufgabe 1:	(39 Punkte)
Aufgabe 2:	(9 Punkte)
Aufgabe 3:	(10 Punkte)
Aufgabe 4:	(9 Punkte)
Aufgabe 5:	(8 Punkte)
Aufgabe 6:	(8 Punkte)
Aufgabe 7:	(7 Punkte)

Gesamtergebnis: .....

Note: .....

1 Multiple Choice (6+5+5+3+4+6+5+5 = 39 Punkte)

Hinweis:
Bei den folgenden Ankreuzaufgaben führen falsche Kreuze zu Punktabzug!
Dabei werden insgesamt jedoch keinesfalls weniger als 0 Punkte für die jeweilige Teilaufgabe vergeben.

(a) Registermaschinen (6 Punkte)

Entscheiden Sie durch Ankreuzen, ob die folgenden Aussagen richtig oder falsch sind.

Die `if-then-else`-Anweisung lässt sich mit der `loop`-Anweisung simulieren.	richtig	☒
	falsch	☐
Jedes LOOP-Programm terminiert.	richtig	☒
	falsch	☐
Es gibt zu jedem WHILE-Programm ein äquivalentes WHILE+IF-Programm, das nur *eine* `while`-Anweisung enthält.	richtig	☒
	falsch	☐
Es gibt zu jedem LOOP-Programm ein äquivalentes WHILE-Programm.	richtig	☒
	falsch	☐
Es gibt zu jedem GOTO-Programm ein äquivalentes LOOP-Programm.	richtig	☐
	falsch	☒
Es gibt zu jeder Turing-Maschine ein äquivalentes GOTO-Programm.	richtig	☒
	falsch	☐

(b) Primitiv rekursive Funktionen (5 Punkte)

Entscheiden Sie durch Ankreuzen, ob die folgenden Aussagen richtig oder falsch sind.

Die Funktion $eq : \mathbb{N}_0 \times \mathbb{N}_0 \to \mathbb{N}_0$ mit $$eq(x,y) = \begin{cases} 1 & \text{falls } x = y \\ 0 & \text{sonst} \end{cases}$$ ist primitiv rekursiv.	richtig ☒ falsch ☐
Jede μ-rekursive Funktion ist primitiv rekursiv.	richtig ☐ falsch ☒
Es gibt zu jedem LOOP-Programm eine äquivalente primitiv rekursive Funktion.	richtig ☒ falsch ☐
Es gibt zu jeder Turing-Maschine eine äquivalente primitiv rekursive Funktion.	richtig ☐ falsch ☒
Die Ackermann-Funktion ist primitiv rekursiv.	richtig ☐ falsch ☒

(c) μ-rekursive Funktionen (5 Punkte)

Entscheiden Sie durch Ankreuzen, ob die folgenden Aussagen richtig oder falsch sind.

Jede μ-rekursive Funktion ist total.	richtig ☐ falsch ☒
Jede totale Funktion ist μ-rekursiv.	richtig ☐ falsch ☒
Die Ackermann-Funktion ist μ-rekursiv.	richtig ☒ falsch ☐
Die Ackermann-Funktion ist total.	richtig ☒ falsch ☐
Es gibt zu jeder μ-rekursiven Funktion ein äquivalentes WHILE-Programm.	richtig ☒ falsch ☐

(d) λ-Kalkül (3 Punkte)

Entscheiden Sie durch Ankreuzen, ob die folgenden Aussagen richtig oder falsch sind.

Zu jeder totalen Funktion gibt es einen λ-Term, der sie berechnet.	richtig ☐ falsch ☒
Es gibt zu jedem WHILE-Programm einen λ-Term, der dieselbe Funktion berechnet.	richtig ☒ falsch ☐
Es gibt zu jedem λ-Term eine Turing-Maschine, die dieselbe Funktion berechnet.	richtig ☒ falsch ☐

(e) Komplexitätstheorie 1 (4 Punkte)

Entscheiden Sie durch Ankreuzen, ob die folgenden Aussagen richtig oder falsch sind.

Sei f eine berechenbare Funktion. Dann ist jede Sprache in $\mathbf{NTIME}(f)$ entscheidbar.	richtig ☒ falsch ☐
Sei f eine berechenbare Funktion. Dann ist jede Sprache in $\mathbf{NSPACE}(f)$ entscheidbar.	richtig ☒ falsch ☐
Sei f eine berechenbare Funktion. Dann gibt es eine natürliche Zahl r, so dass gilt: $\mathbf{NTIME}(f) \subseteq \mathbf{DTIME}(r^f)$	richtig ☒ falsch ☐
Sei f eine berechenbare Funktion. Dann gilt: $\mathbf{DSPACE}(f) \subseteq \mathbf{DTIME}(f)$	richtig ☐ falsch ☒

(f) Komplexitätstheorie 2 (6 Punkte)

Entscheiden Sie durch Ankreuzen, ob die folgenden Aussagen richtig oder falsch sind.

Jedes Problem in **NP** ist **NP**-hart.	richtig	☐
	falsch	☒
Wenn es ein **NP**-vollständiges Problem gibt, das in **P** liegt, dann gilt **P** = **NP**.	richtig	☒
	falsch	☐
Aus **P** = **co-P** folgt sofort auch **NP** = **co-NP**.	richtig	☐
	falsch	☒
Wenn L **NP**-vollständig ist, dann ist $\overline{L}$ **co-NP**-vollständig.	richtig	☒
	falsch	☐
Wenn es ein Problem gibt, das sowohl **NP**-vollständig als auch **co-NP**-vollständig ist, dann gilt **NP** = **co-NP**.	richtig	☒
	falsch	☐
PSPACE = **NPSPACE**	richtig	☒
	falsch	☐

(g) Komplexitätstheorie 3 (5 Punkte)

Entscheiden Sie durch Ankreuzen, ob die folgenden Aussagen richtig oder falsch sind.

Seien L_1 und L_2 Sprachen und es gelte $L_1 \preceq_{pol} L_2$ und $L_2 \in$ **NP**. Dann gilt auch $L_1 \in$ **NP**.	richtig	☒
	falsch	☐
Das Hamilton-Kreis-Problem ist **NP**-vollständig.	richtig	☒
	falsch	☐
3-*CLIQUE* ist **NP**-vollständig.	richtig	☐
	falsch	☒
3-*DNF-SAT* ist **NP**-vollständig.	richtig	☐
	falsch	☒
Ein **NP**-vollständiges Problem heißt pseudopolynomiell, wenn es eine Turing-Maschine gibt, die in polynomieller Zeit Näherungslösungen für das Problem berechnet.	richtig	☐
	falsch	☒

(h) Entscheidbarkeit, Aufzählbarkeit, Satz von Rice (5 Punkte)

Entscheiden Sie durch Ankreuzen, ob die folgenden Aussagen richtig oder falsch sind.

Die Frage, ob ein gegebenes WHILE-Programm dieselbe Funktion wie ein gegebener λ-Term berechnet, ist entscheidbar.	richtig ☐ falsch ☒
Die Frage, ob ein gegebenes WHILE-Programm das Produkt zweier Zahlen berechnet, ist entscheidbar.	richtig ☐ falsch ☒
Die Menge aller WHILE-Programme, die das Produkt zweier Zahlen berechnen, ist aufzählbar.	richtig ☐ falsch ☒
Die Frage, ob ein gegebenes Java-Programm für eine gegebene Eingabe bei der Ausführung eine NullPointer-Exception wirft, ist entscheidbar.	richtig ☐ falsch ☒
Es gibt Java-Programme, für die man mit Sicherheit sagen kann, dass sie für keine Eingabe bei der Ausführung eine NullPointer-Exception werfen.	richtig ☒ falsch ☐

2 Registermaschinen 1 (3+6 = 9 Punkte)

Ein GOTO-Befehl (im Sinne dieser Aufgabe) hat eine der folgenden Formen:
- x_i `:=` c
- x_i `:=` x_j
- x_i `:=` c *op* x_j
- x_i `:=` x_j *op* c
- x_i `:=` x_j *op* x_k
- `goto` l
- `if` x_i `= 0 goto` l

Dabei sind
x_i, x_j Register,
c ist eine Konstante aus $\mathbb{N}_0$,
$op \in \{+, \dot{-}, *\}$,
und l ist ein Label.

Ein GOTO-Programm (im Sinne dieser Aufgabe) hat die Form

$$l_1 : B_1; \ldots; l_k : B_k \qquad (k \geq 1),$$

dabei sind $B_1, \ldots, B_k$ GOTO-Befehle und $l_1, \ldots, l_k$ paarweise verschiedene Labels.

Die Funktion $sqrt : \mathbb{N}_0 \to \mathbb{N}_0$ sei wie folgt definiert:

$$sqrt(n) = \lfloor \sqrt{n} \rfloor$$

(a) Geben Sie ein Pseudocode-Programm für *sqrt* an.

Dieses darf nur die arithmetischen Operatoren $+, \dot{-}, *$ und Vergleichsoperatoren wie $<, \leq, \ldots$ verwenden. Ansonsten muss es *nicht* den oben genannten Formvorgaben für GOTO-Programme genügen.

Lösung:

```
w := 0;
while (w + 1) * (w + 1) ≤ n do
  w := w + 1
end;
output w
```

(b) Geben Sie nun ein GOTO-Programm für *sqrt* an.

Beachten Sie dabei auch, dass alle Register außer dem Ergebnis-Register am Ende der Programmausführung dieselben Werte wie am Anfang enthalten müssen.

Lösung:

```
 1 : x2 := 0;
 2 : x3 := x2 + 1;
 3 : x3 := x3 * x3;
 4 : x3 := x3 ∸ x1;
 5 : x3 := 1 ∸ x3;
 6 : if x3 = 0 goto 13;
 7 :    x2 := x2 + 1;
 8 :    x3 := x2 + 1;
 9 :    x3 := x3 * x3;
10 :    x3 := x3 ∸ x1;
11 :    x3 := 1 ∸ x3;
12 : goto 6;
13 : x3 := 0
```

3 Registermaschinen 2 (8+1+1 = 10 Punkte)

Gegeben sei das folgende LOOP-Programm P:

```
x3 := 1;
// (1)
loop x2 do
  x4 := 0;
  loop x3 do
    x4 := x4 + x1
  end;
  x3 := x4
end;
// (2)
x4 := 0
```

(a) Füllen Sie die folgende Tabelle mit den Werten der Register x_1, x_2, x_3 an den beiden Programmstellen (1) und (2), und zwar jeweils

i. für die Eingabe $x_1 = 2, x_2 = 3$ und

ii. für die Eingabe $x_1 = 3, x_2 = 2$.

Eingabe i.	x_1	x_2	x_3
(1)	2	3	1
(2)	2	3	8

Eingabe ii.	x_1	x_2	x_3
(1)	3	2	1
(2)	3	2	9

(b) Welche Werte berechnet P für folgende Eingaben?

i. Eingabe: $x_1 = 2, x_2 = 3$ – Ausgabe: 8.....

ii. Eingabe: $x_1 = 3, x_2 = 2$ – Ausgabe: 9.....

(c) Welche Funktion $f : \mathbb{N}_0 \times \mathbb{N}_0 \to \mathbb{N}_0$ berechnet P?

$$f(x, y) \quad = \quad x^y \text{.....}$$

4 μ-rekursive Funktionen (3+6 = 9 Punkte)

Der beschränkte Maximums-Operator sei für

$$f : \mathbb{N}_0^{k+1} \to \mathbb{N}_0 \qquad \text{und} \qquad t \in \mathbb{N}_0$$

wie folgt definiert:

$$\max_{i \leq t}(f(\vec{n}, i) = 0) \;=\; \begin{cases} \max\{i \in \mathbb{N}_0 \mid i \leq t \wedge f(\vec{n}, i) = 0\} & \text{falls das Maximum existiert} \\ \textit{undefiniert} & \text{sonst} \end{cases}$$

(a) Zeigen Sie:

Ist f μ-rekursiv, so ist auch die Funktion $g : \mathbb{N}_0^{k+1} \to \mathbb{N}_0$ mit

$$g(\vec{n}, t) = \max_{i \leq t}(f(\vec{n}, i) = 0)$$

μ-rekursiv.

Lösung:
Der max-Operator lässt sich mit Hilfe des μ-Operators darstellen:

$$g(\vec{n}, t) \;=\; \mu i(f(\vec{n}, t \dot{-} i) = 0)$$

(b) Zeigen Sie mit Hilfe des beschränkten Maximums-Operators, dass die Funktion

$$\begin{array}{l} \textit{ggt} \;:\; \mathbb{N}_0 \times \mathbb{N}_0 \to \mathbb{N}_0 \qquad \text{mit} \\ \textit{ggt}(x, y) \;=\; \text{der größte gemeinsame Teiler von } x \text{ und } y \end{array}$$

μ-rekursiv ist.

Dabei dürfen Sie alle Funktionen als μ-rekursiv voraussetzen, für die dies in der Vorlesung oder auf einem Aufgabenblatt schon bewiesen wurde, und zudem die folgende Funktion:

$$\textit{teilt} \;:\; \mathbb{N}_0 \times \mathbb{N}_0 \to \mathbb{N}_0 \qquad \text{mit}$$

$$\textit{teilt}(n, m) = \begin{cases} 1 & \text{falls } n \neq 0 \text{ und } n|m \text{ („}n \text{ teilt } m\text{“)} \\ 0 & \text{sonst} \end{cases}$$

Lösung:
Der ggT von x und y ist immer kleiner als oder gleich $\min(x, y)$. Die Funktion min ist μ-rekursiv, denn

$$\min(x, y) = y \dot{-} (y \dot{-} x)$$

(Dank an Viktor Seib, dem diese Lösung eingefallen ist.)
Damit gilt nun:

$$\textit{ggt}(x, y) = \max_{i \leq \min(x,y)}(1 \dot{-} (\textit{teilt}(i, x) * \textit{teilt}(i, y)) = 0)$$

5 Primitiv rekursive Funktionen (8 Punkte)

Die Funktion $d : \mathbb{N}_0 \to \mathbb{N}_0$ sei wie folgt definiert:

$$d(n) = \text{Anzahl der Teiler von } n$$

Es gilt also

$$d(0) = 0, \quad d(1) = 1, \quad d(2) = 2, \quad d(3) = 2, \quad d(4) = 3, \quad d(12) = 6 \quad \text{usw.}$$

Zeigen Sie, dass d primitiv rekursiv ist.

Dabei dürfen Sie alle Funktionen verwenden, von denen in der Vorlesung oder auf einem Aufgabenblatt schon bewiesen wurde, dass sie primitiv rekursiv sind. Außerdem dürfen Sie die folgende Funktion verwenden:

$$\begin{aligned} &\mathit{teilt} \ : \ \mathbb{N}_0 \times \mathbb{N}_0 \to \mathbb{N}_0 \\ &\mathit{teilt}(n,m) = \begin{cases} 1 & \text{falls } n \neq 0 \text{ und } n|m \ (\text{„}n \text{ teilt } m\text{“}) \\ 0 & \text{sonst} \end{cases} \end{aligned}$$

Zum Beweis genügt es, dass Sie Lücken in den folgenden Definitionen der Funktion d und der Hilfsfunktion d_2 ausfüllen:

$$\begin{aligned} d(n) &= d_2(n,n) \\ d_2(n,0) &= \underline{\quad 0 \quad} \\ d_2(n,m+1) &= \underline{\quad \mathit{teilt}(m+1,n) + d_2(n,m) \quad} \end{aligned}$$

6 λ-Kalkül (8 Punkte)

Im Folgenden bezeichnet $\overline{n}$ den λ-Term, der eine Zahl $n \in \mathbb{N}_0$ repräsentiert.

Geben Sie einen λ-Term a an, so dass für alle $n \in \mathbb{N}_0$ gilt:

$$a\,\overline{n} \equiv \begin{cases} \bar{1} & \text{falls } n = 0 \\ \bar{2} & \text{falls } n = 1 \\ \bar{4} & \text{falls } n = 2 \\ a\,\overline{(n-1)} + a\,\overline{(n-2)} - a\,\overline{(n-3)} & \text{sonst} \end{cases}$$

Dabei dürfen Sie alle λ-Terme verwenden, die bisher in der Vorlesung oder auf einem Aufgabenblatt definiert wurden – auch den Fixpunkt-Kombinator Y.

Außerdem dürfen Sie die λ-Terme $=$ und $\dot{-}$ verwenden, für die für alle $x, y \in \mathbb{N}_0$ gilt:

$$\begin{aligned} \overline{x} = \overline{y} &\equiv \begin{cases} \mathit{true} & \text{falls } x = y \\ \mathit{false} & \text{sonst} \end{cases} \\ \overline{x} \dot{-} \overline{y} &\equiv \overline{x \dot{-} y} \end{aligned}$$

Lösung:

$$a \;=_{\text{def}}\; Y\,F$$

mit

$$\begin{aligned} F \;=_{\text{def}}\; \lambda f\,.\,\lambda n\,.\; & \mathit{if}\ n = \bar{0}\ \mathit{then}\ \bar{1}\ \mathit{else} \\ & \mathit{if}\ n = \bar{1}\ \mathit{then}\ \bar{2}\ \mathit{else} \\ & \mathit{if}\ n = \bar{2}\ \mathit{then}\ \bar{4}\ \mathit{else} \\ & f(n \dot{-} \bar{1}) + f(n \dot{-} \bar{2}) \dot{-} f(n \dot{-} \bar{3}) \end{aligned}$$

7 Komplexitätstheorie (5+2 = 7 Punkte)

Für jede natürliche Zahl k sei k-*CNF-SAT* definiert als das aussagenlogische Erfüllbarkeitsproblem für solche Klauselmengen, in denen jede Klausel höchstens k Literale hat.

(a) Zeigen Sie:

4-*CNF-SAT* lässt sich polynomiell auf 3-*CNF-SAT* reduzieren.

Es genügt, die Konstruktion anzugeben, mit der ein Problem aus 4-*CNF-SAT* in ein Problem aus 3-*CNF-SAT* übersetzt werden kann; der Beweis der Korrektheit ist nicht notwendig. Sie müssen auch nicht beweisen, dass die Konstruktion in polynomieller Zeit möglich ist.

Lösung:

Sei K ein beliebiges Problem aus 4-*CNF-SAT*. Dieses wird wie folgt in ein Problem K' aus 3-*CNF-SAT* übersetzt:

- Jede Klausel aus K, die weniger als vier Literale hat, wird unverändert in K' übernommen.
- Für jede Klausel
$$C = \{L_1, L_2, L_3, L_4\} \in K$$
mit vier Literalen werden die folgenden zwei Klauseln mit drei Literalen in K' aufgenommen, wobei X_C eine neue, bisher nicht vorkommende aussagenlogische Variable ist:
$$\{L_1, L_2, X_C\}, \qquad \{L_3, L_4, \neg X_C\}$$

(b) Kann man auf dieselbe Art auch Folgendes zeigen?

3-*CNF-SAT* lässt sich polynomiell auf 2-*CNF-SAT* reduzieren.

Begründen Sie Ihre Antwort.

Lösung:

Nein, das ist nicht möglich.
Mit der angegebenen Transformation lässt sich im allgemeinen ein Problem aus k-*CNF-SAT* pynomiell auf ein k'-*CNF-SAT* Problem reduzieren mit

$$k' = \lceil k/2 \rceil + 1$$

Für $k = 4$ gilt $k' = \lceil 4/2 \rceil + 1 = 3$ und man erreicht das Gewünschte.

Für $k = 3$ dagegen gilt $k' = \lceil 3/2 \rceil + 1 = 3 \neq 2$, und die Transformation liefert nicht das Gewünschte.