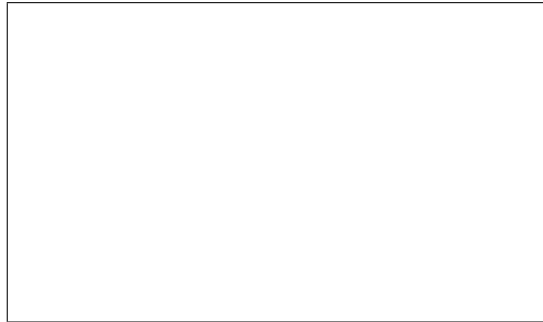


# Musterlösung Informatik I

## 23.04.2003

Prof. Dr. G. Goos  
Dipl.-Inform. M. L. Noga



Zur Klausur sind keine Hilfsmittel und kein eigenes Papier zugelassen. Die Bearbeitungszeit beträgt 60 Minuten.

Die Klausur ist komplett und geheftet abzugeben. **Nur Blätter, die Namen und Matrikelnummer tragen, gehen in die Bewertung ein.**

Sie dürfen die Rückseite der Aufgabenblätter als Konzeptpapier benutzen. **Nur Lösungen, die sich auf dem entsprechenden Aufgabenblatt oder der vorangehenden Rückseite befinden, gehen in die Bewertung ein.**

Aufgabe	1	2	3	4	5	6	$\Sigma$
Maximal	10	10	9	10	12	9	60
K1							
K2							
K3							

Bonus:

Note:

## Aufgabe 1: Wissen (10 Punkte)

Welche der folgenden Aussagen sind wahr (  *W* ) und welche falsch (  *F* )? Kreuzen Sie an. Jedes korrekte Kreuz zählt 0,5 Punkte, jedes falsche Kreuz kostet 0,5 Punkte Abzug. Diese Aufgabe wird mindestens mit 0 Punkten bewertet.

1.  *W*  *F* Jeder deterministische Algorithmus terminiert.
2.  *W*  *F* Semi-Thue-Systeme sind deterministisch.
3.  *W*  *F* Markov-Algorithmen sind deterministisch.
4.  *W*  *F* Zwei schwach äquivalente Grammatiken definieren dieselbe Sprache.
5.  *W*  *F* Jeder Verband ist eine boolesche Algebra.
6.  *W*  *F* Jeder boolesche Algebra ist ein Verband.
7.  *W*  *F* Für jede Halbordnung und beliebige  $x, y$  gilt  $x \leq y \vee y \leq x$
8.  *W*  *F* Alle lokal konfluenten noetherschen Halbordnungen sind konfluent.
9.  *W*  *F* In der Aussagenlogik ist jeder semantische Schluß syntaktisch ableitbar.
10.  *W*  *F* Die konjunktive Normalform ist eine Disjunktion von Konjunktionen.
11.  *W*  *F* Gödel bewies die Unvollständigkeit der Prädikatenlogik 1. Ordnung.
12.  *W*  *F* Der  $\lambda$ -Kalkül ist konfluent.
13.  *W*  *F* Für beliebige  $f$  und  $g$  gilt  $\text{map } f (\text{map } g \text{ xs}) = \text{map } (g.f) \text{ xs}$ .
14.  *W*  *F* Ein Programm, welches unbeschränkte Listen verwendet, kann terminieren.
15.  *W*  *F* Alle Keller sind LIFOs.
16.  *W*  *F* Alle Prioritätsschlangen sind FIFOs.
17.  *W*  *F* Binäre Suche auf Arrays hat den Aufwand  $O(\log(n))$ .
18.  *W*  *F* Die Rekurrenz  $f(n) = O(n) + 2 * f(n/3)$  hat den Aufwand  $O(n)$ .
19.  *W*  *F* Die Rekurrenz  $f(n) = O(n) + 3 * f(n/2)$  hat den Aufwand  $O(n)$ .
20.  *W*  *F* Gierige Algorithmen arbeiten ohne Rücksetzen.

	K1	K2	K3
<b>A1</b>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

## Aufgabe 2: Formale Grundlagen (4+3+3=10 Punkte)

Korrekte Kreuze zählen 0,5 Punkte, falsche Kreuze kosten 0,5 Punkte Abzug. Jede Teilaufgabe wird mindestens mit 0 Punkten bewertet.

a) Sei  $G = (\Sigma, N, P, S)$  mit  $\Sigma = \{a, b, c, e, h, n, r, s, t, w\}$ ,  $N = \{S, X, Y, Z\}$  und

$$P = \left\{ \begin{array}{llll} S & \rightarrow X, & & \\ X & \rightarrow Ye & | Yt & | Yen \\ Y & \rightarrow hZ & | trZ & | schwZ \\ Z & \rightarrow ab & & \end{array} \right\}$$

Welche Sprache erzeugt diese Grammatik? Geben Sie den engsten Chomsky-Typ dieser Grammatik an. Unabhängig von der Grammatik, welches ist der engste Chomsky-Typ der Sprache?

$$L(G) = \{habe, habt, haben, trabe, trabt, traben, schwabe, schwabt, schwaben\}$$

$$G \text{ liegt in } \begin{array}{cccc} \boxed{CH-0} & \boxed{CH-1} & \boxed{CH-2} & \boxed{CH-3} \end{array}$$

$$L(G) \text{ liegt in } \begin{array}{cccc} \boxed{CH-0} & \boxed{CH-1} & \boxed{CH-2} & \boxed{CH-3} \end{array}$$

b) Sei  $L_{PG}$  die Sprache der Palindrome gerader Länge über  $\Sigma = \{a, b\}$ . Geben Sie die engste Chomsky-Klasse an, welche  $L_{PG}$  enthält. Geben Sie eine Grammatik  $G$  dieser Klasse mit  $L(G) = L_{PG}$  an. Vermeiden Sie dabei redundante Nichtterminale und Produktionen.

$$L_{PG} \text{ liegt in } \begin{array}{cccc} \boxed{CH-0} & \boxed{CH-1} & \boxed{CH-2} & \boxed{CH-3} \end{array}$$

$$G = (\Sigma, N, P, S) \text{ mit } N = \{S\} \text{ und}$$

$$S = \{S \rightarrow aSa \mid bSb \mid \epsilon\}$$

c) Gegeben sei eine Darstellung von Binärzahlen als Worte über  $\Sigma = \{0, 1\}$ , wobei das höchstwertigste Bit links steht. Schreiben Sie einen gesteuerten Markov-Algorithmus, welcher das Zweierkomplement einer gegebenen Zahl fester Breite berechnet.

$$\begin{array}{l} 1\beta \rightarrow \beta 0 \\ 0\beta \rightarrow 1. \\ \beta \rightarrow . \\ \alpha 1 \rightarrow 0\alpha \\ \alpha 0 \rightarrow 1\alpha \\ \alpha \rightarrow \beta \\ \epsilon \rightarrow \alpha \end{array}$$

**A2**

K1

K2

K3

### Aufgabe 3: SQL (3+3+3=9 Punkte)

Sie sind Juniorberater bei Klugschwätz & Partner. Ihr erster Kunde ist die Eisenklopfer AG, die ihre Personaldaten in folgenden Tabellen unterhält (Primärschlüssel sind fettgedruckt):

**Personal**

Feld	Typ
<b>persNr</b>	int
name	String
gehalt	int

**Abteilungen**

Feld	Typ
<b>abtNr</b>	int
name	String
ortNr	int

**ArbeitetIn**

Feld	Typ
<b>persNr</b>	int
<b>abtNr</b>	int
leitet	Bool

**Standorte**

Feld	Typ
<b>ortNr</b>	int
name	String
maxGehalt	int

a) Prüfen Sie folgende Aussagen über den Kunden. Korrekte Kreuze zählen 0,5 Punkte, falsche Kreuze kostet 0,5 Punkte Abzug. Diese Teilaufgabe wird mindestens mit 0 Punkten bewertet.

- W*    *F*   Jeder Mitarbeiter arbeitet in höchstens einer Abteilung  
 *W*    *F*   Ein Mitarbeiter kann an mehreren Standorten tätig sein  
 *W*    *F*   Eine Abteilung kann mehrere Leiter aufweisen  
 *W*    *F*   Jede Abteilung hat höchstens einen Standort  
 *W*    *F*   Kein Mitarbeiter kann mehr verdienen als sein Abteilungsleiter  
 *W*    *F*   Kein Abteilungsleiter ist einem anderen Abteilungsleiter untergeordnet

b) Klugschwätz propagiert niedrige Betreuungsquotienten als Unternehmensziel. Ihr Statistikprogramm erwartet eine Liste der Paare von Abteilungsleitern und jeweiligen Untergebenen. Erstellen Sie eine SQL-Anfrage, welche die entsprechenden Personalnummern ausgibt.

```

SELECT DISTINCT L.persNr, MA.persNr
FROM   ArbeitetIn as L, ArbeitetIn as MA
WHERE  L.leitet = TRUE      AND
       L.abtNr  = MA.abtNr AND
       MA.leitet = FALSE
// Co-Abteilungsleiter sind keine Untergebenen!
  
```

c) Ihr Seniorberater empfiehlt dem Vorstand der Eisenklopfer einen harten Sparkurs. Er hat für viele Standorte neue Maximalgehälter eingetragen. Schreiben Sie eine SQL-Anfrage, welche die Personalnummern der redundanten, d.h. freizusetzenden, Mitarbeiter ermittelt.

	K1	K2	K3
<b>A3</b>	<input type="text"/>	<input type="text"/>	<input type="text"/>

```
SELECT DISTINCT persNr
FROM ((Personal INNER JOIN ArbeitetIn ON Personal.persNr =ArbeitetIn.persNr
      )
      INNER JOIN Abteilungen ON ArbeitetIn.abtNr =Abteilungen.abtNr
      )
      INNER JOIN Standorte ON Abteilungen.ortNr=Standorte.ortNr
WHERE  gehalt > maxGehalt

-- Variante
SELECT DISTINCT Personal.persNr
FROM Personal, Abteilungen, ArbeitetIn, Standorte
WHERE Personal.gehalt > Standorte.maxGehalt
AND Personal.persNr = ArbeitetIn.persNr
AND Abteilungen.abtNr = ArbeitetIn.persNr
AND Abteilungen.ortNr = Standort.ortNr
```

**A3**

K1

K2

K3

### Aufgabe 4: Formale Logik (4+6=10 Punkte)

a) Ein Volladdierer berechnet aus Eingangsbits  $x, y$  und Eingangsübertrag  $c$  den Übertrag  $c' = x \wedge y \vee x \wedge c \vee y \wedge c$  und das Summenbit  $s = x \leftrightarrow y \leftrightarrow c$ . Leider kann unser  $\tau 2logic^{\text{®}}$  Simulator nur DNF verarbeiten. Formen Sie  $s = x \leftrightarrow y \leftrightarrow c$  in diese Darstellung um.

$$\begin{aligned}
 s &= x \leftrightarrow y \leftrightarrow c \\
 &= (x \wedge y \vee \bar{x} \wedge \bar{y}) \leftrightarrow c \\
 &= (x \wedge y \vee \bar{x} \wedge \bar{y}) \wedge c \vee \neg(x \wedge y \vee \bar{x} \wedge \bar{y}) \wedge \bar{c} \\
 &= x \wedge y \wedge c \vee \bar{x} \wedge \bar{y} \wedge c \vee \neg(x \wedge y \vee \bar{x} \wedge \bar{y}) \wedge \bar{c} \\
 &= x \wedge y \wedge c \vee \bar{x} \wedge \bar{y} \wedge c \vee ((\bar{x} \vee \bar{y}) \wedge (x \vee y)) \wedge \bar{c} \\
 &= x \wedge y \wedge c \vee \bar{x} \wedge \bar{y} \wedge c \vee (\bar{x} \wedge (x \vee y) \vee \bar{y} \wedge (x \vee y)) \wedge \bar{c} \\
 &= x \wedge y \wedge c \vee \bar{x} \wedge \bar{y} \wedge c \vee (\bar{x} \wedge x \vee \bar{x} \wedge y \vee \bar{y} \wedge x \vee \bar{y} \wedge y) \wedge \bar{c} \\
 &= x \wedge y \wedge c \vee \bar{x} \wedge \bar{y} \wedge c \vee (\bar{x} \wedge y \vee \bar{y} \wedge x) \wedge \bar{c} \\
 &= x \wedge y \wedge c \vee \bar{x} \wedge \bar{y} \wedge c \vee \bar{x} \wedge y \wedge \bar{c} \vee x \wedge \bar{y} \wedge \bar{c}
 \end{aligned}$$

b) Gegeben sei die prädikatenlogische Formel  $F = \neg \exists z : P(z) \wedge \forall y : Q(y) \rightarrow \forall x : R(x, y, z)$ . Bereinigen Sie zunächst die Operatoren. Stellen Sie dann die Pränexform her, und bringen Sie den quantorenfreien Teil der Formel in DNF. Erstellen Sie danach die bereinigte Skolemform.

$$\begin{aligned}
 F &= \neg \exists z : P(z) \wedge \forall y : Q(y) \rightarrow \forall x : R(x, y, z) \\
 &= \neg \exists z : P(z) \wedge \forall y : \overline{Q(y)} \vee \forall x : R(x, y, z) \quad (\text{bereinigt}) \\
 &= \forall z : \neg [P(z) \wedge \forall y : \overline{Q(y)} \vee \forall x : R(x, y, z)] \\
 &= \forall z : \neg \forall y : [P(z) \wedge (\overline{Q(y)} \vee \forall x : R(x, y, z))] \\
 &= \forall z : \exists y : \neg [P(z) \wedge (\overline{Q(y)} \vee \forall x : R(x, y, z))] \\
 &= \forall z : \exists y : \neg \forall x : [P(z) \wedge (\overline{Q(y)} \vee R(x, y, z))] \\
 &= \forall z : \exists y : \exists x : \neg [P(z) \wedge (\overline{Q(y)} \vee R(x, y, z))] \quad (\text{bereinigte Pränexform}) \\
 &= \forall z : \exists y : \exists x : [\overline{P(z)} \vee \neg(\overline{Q(y)} \vee R(x, y, z))] \\
 &= \forall z : \exists y : \exists x : [\overline{P(z)} \vee (Q(y) \wedge \overline{R(x, y, z)})] \quad (\text{DNF}) \\
 &= \forall z : [\overline{P(z)} \vee (Q(sk_1(z)) \wedge \overline{R(sk_2(z), sk_1(z), z)})] \quad (\text{bereinigte Skolemform})
 \end{aligned}$$

A4

K1

K2

K3

## Aufgabe 5: Haskell (5+4+3=12 Punkte)

a) Seien  $A = a_1, \dots, a_n$  und  $B = b_1, \dots, b_m$  zwei endliche Folgen, potentiell leer.  $B$  ist eine Teilfolge von  $A$  wenn  $\exists i : a_{i+1}, \dots, a_{i+m} = B$ . Der ADT *endliche Folge* sei als Liste umgesetzt.

Implementieren Sie die Operation `contains :: Eq a => [a] -> [a] -> Bool`, welche prüft, ob ihr zweites Argument eine Teilfolge des ersten ist.

```
-- Mit Bibliothek (weniger effizient)
contains f tf | lf < lt      = False
              | take lt f == tf = True
              | otherwise    = contains (tail f) tf
  where lf = length f
        lt = length tf

-- Mit Hilfsfunktion (effizienter)
contains f      [] = True
contains []    tf = False
contains (f:fs) tf = headEqual (f:fs) tf || contains fs tf

headEqual f      []      = True
headEqual []    tf      = False
headEqual (f:fs) (t:tfs) = f == t && headEqual fs tfs
```

b) Functional Solutions Inc. vertreibt eine Branchenlösung in Haskell mit 800.000 Zeilen Code. Leider ist das System zu langsam. Es verbringt 90% seiner Rechenzeit in folgender Funktion:

```
filter2 f xs = (filter f xs, filter (not.f) xs)
```

Zur Veranschaulichung: `filter2 even [1..4]` liefert `([2,4],[1,3])`.

Schreiben Sie `filter2` so um, daß höchstens ein Abstieg in die Eingabeliste erfolgt.

```
filter2 _ [] = ( [], [] )
filter2 p (x:xs) | p x = (x:ts, fs)
                  | otherwise = ( ts, x:fs)
  where (ts,fs) = filter2 p xs

-- Variante mit Durchreichen von Teilergebnissen
filter2 f xs = filter2' f (reverse xs) [] []
filter2' _ [] (ts,fs) = (ts,fs)
filter2' f (x:xs) (ts,fs) | f x = filter2' f xs (x:ts, fs)
                           | otherwise = filter2' f xs ( ts, x:fs)

-- Variante mit Funktionen höherer Ordnung
filter2 f xs = foldr combine ([],[ ]) (map (pairgen f) xs)
  where pairgen f x | f x = ([x],[ ])
                  | otherwise = ([ ],[x])
        combine (l,r) (l',r') = (l++l',r++r')
```

c) Gegeben eine Liste  $l :: [a]$ , so bezeichnen wir die Anzahl der Vorkommnisse eines Elements in  $l$  als seine Kardinalität. Die Liste  $k :: [(a, \text{Int})]$  von Elementen und Kardinalitäten sei nach dem ersten Auftreten des Elements in  $l$  geordnet. Für  $l = \text{"hallowelt"}$  gilt z.B.

	K1	K2	K3
<b>A5</b>	<input type="text"/>	<input type="text"/>	<input type="text"/>

$k = [('h',1), ('a',1), ('l',3), ('o',1), ('w',1), ('e',1), ('t',1)]$ .

Schreiben Sie eine Funktion `cards :: Eq a => [a] -> [(a,Int)]`, welche `k` aus `l` berechnet. Sie dürfen `filter2` auch dann verwenden, wenn Sie Teilaufgabe b) nicht gelöst haben.

```
cards []      = []
cards (x:xs) = (x, 1 + length ts) : cards fs
              where (ts,fs) = filter2 (==x) xs
```

**A5**

K1

K2

K3

## Aufgabe 6: Terminierung (3+1+2+3=9 Punkte)

Gegeben sei folgende Funktion auf positiven natürlichen Zahlen  $a_0, b_0$  und  $c_0$ :

```
func a0 b0 c0
= func' a0 b0 c0 where
  func' a b c | a==b && b==c = a
              | a< b       = func' (a+a0) b c
              | b< c       = func' a (b+b0) c
              | otherwise   = func' a b (c+c0)
```

a) Berechnen Sie `func 2 6 10`.

30

b) Wenn `func` terminiert, welche mathematische Funktion berechnet sie?

$kgV(a_0, b_0, c_0)$

c) Zeigen Sie: Für jeden Aufruf  $i$  von `func'` gibt es ein  $k_i \in \mathbb{N}$  mit  $a_i = k_i a_0$ . Die Folge dieser  $k_i$  wächst monoton um 0 oder 1.

Induktionsbeginn: Initial gilt  $k_0 = 1$ .

Induktionsschritt: Gilt aktuell  $a_i = k_i a_0$ , so unterscheiden wir drei Fälle. Ist  $a_i = b_i = c_i$  erfolgen keine weiteren Aufrufe. Ist  $a_i < b_i$ , so gilt die Aussage im nächsten Schritt für  $k_{i+1} = k_i + 1$ , ansonsten für  $k_{i+1} = k_i$ .

Damit gilt die Aussage für alle Aufrufe.

d) Es gelte die Aussage der vorigen Teilaufgabe sowie analoge Aussagen für die  $b_i$  und  $c_i$ . Zeigen Sie: `func` terminiert. Dazu ist zu zeigen: die  $a_i, b_i$  und  $c_i$  sind durch  $s = a_0 b_0 c_0$  beschränkt.

Induktionsbeginn: Initial gilt  $a_0 \leq s, b_0 \leq s, c_0 \leq s$ .

Induktionsschritt: Gilt  $a_i = b_i = c_i = s$ , so terminiert die Funktion. Ansonsten wird genau eine Variable erhöht. Diese muß echt kleiner als eine andere Variable sein. ObdA sei  $a_i < b_i$ .

$$\begin{aligned}
 a_i < b_i &\Rightarrow^{\text{IV}} \\
 a_i < s &\Rightarrow^{\text{Voraussetzung}} \\
 k_i a_0 < s &\Leftrightarrow \\
 k_i < b_0 c_0 &\Rightarrow^{\text{Voraussetzung}} \\
 k_i + 1 \leq b_0 c_0 &\Leftrightarrow \\
 a_{i+1} \leq s &
 \end{aligned}$$

Damit gilt die Aussage für alle Aufrufe.

	K1	K2	K3
<b>A6</b>	<input type="text"/>	<input type="text"/>	<input type="text"/>