

Musterlösung Informatik I

20.02.2003

Prof. Dr. G. Goos

Dipl.-Inform. M. L. Noga

Vorname:

Name:

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 seiner Rückseite befinden, gehen in die Bewertung ein.**

Zum Bestehen der Klausur waren 16 Punkte nötig.

Aufgabe	1	2	3	4	5	6	Σ
Maximal	10	10	9	10	11	10	60
K1							
K2							
K3							

Punkte:

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. Es gibt zwei Darstellungen der Null bei Verwendung
 - W* *F* von Vorzeichen und Betrag
 - W* *F* des Einerkomplements für negative Zahlen
 - W* *F* des Zweierkomplements für negative Zahlen
2. Es gibt ein kleinstes Element in jeder
 - W* *F* halbgeordneten Menge
 - W* *F* vollständig halbgeordneten Menge
 - W* *F* total geordneten Menge
3. *W* *F* Jeder Baum ist schwach zusammenhängend.
4. *W* *F* Jeder schwach zusammenhängende Graph ist ein Baum.
5. *W* *F* Jeder bipartite Graph ist unzusammenhängend.
6. *W* *F* Jedes Petrinetz hat eine konstante Anzahl Stellen.
7. *W* *F* Jedes lebendige Petrinetz hat eine monoton wachsende Anzahl Marken.
8. *W* *F* $\lambda x.(\lambda y.z)$ ist ein Churchscher Wahrheitswert.
9. *W* *F* Der λ -Kalkül arbeitet mit α -, β - und γ -Konversionen.
10. Eine Ordnung auf den Elementen ist Voraussetzung für den ADT
 - W* *F* Schlange
 - W* *F* Prioritätsschlange
 - W* *F* Binärer Suchbaum
11. Ohne Verteilungsannahme enthält die Aufwandsklasse $O(n \log(n))$
 - W* *F* Sortieren durch Einfügen (Insertion sort)
 - W* *F* Sortieren durch Mischen (Mergesort)
 - W* *F* Sortieren durch Zerlegen (Quicksort)
12. *W* *F* Binäre Suche auf Listen ist $O(\log(n))$.

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

Aufgabe 2: Formale Sprachen (2+3+5=10 Punkte)

Sei $G_i = (\Sigma, N, P_i, S)$ mit $\Sigma = \{a, b, c, d\}$, $N = \{A, B, C, D, S\}$ und

$$\begin{array}{lll}
 P_1 = \{S \rightarrow A, & P_2 = \{S \rightarrow ABC, & P_3 = \{S \rightarrow aA, \\
 A \rightarrow BC, & AB \rightarrow a, & A \rightarrow bB \mid cC, \\
 B \rightarrow bBb \mid D, & BC \rightarrow a, & B \rightarrow aA, \\
 C \rightarrow cCc \mid D, & aC \rightarrow aaCa \mid c, & C \rightarrow aA \mid dD, \\
 D \rightarrow d & Aa \rightarrow aAaa \mid c \} & D \rightarrow d & \}
 \end{array}$$

Geben Sie jeweils den engsten Chomsky-Typ der Grammatik, die von der Grammatik erzeugte Sprache sowie den engsten Chomsky-Typ dieser Sprache an.

Jedes korrekte Kreuz zählt 0,5 Punkte, jedes falsche Kreuz kostet 0,5 Punkte Abzug. Diese Aufgabe wird mindestens mit 0 Punkten bewertet.

G_1 liegt in CH-0 CH-1 CH-2 CH-3

$L(G_1) = b^n db^n c^m dc^m$ mit $m, n \geq 0$.

$L(G_1)$ liegt in CH-0 CH-1 CH-2 CH-3

G_2 liegt in CH-0 CH-1 CH-2 CH-3

$L(G_2) = a^n ca^n$ mit $n \geq 0$.

$L(G_2)$ liegt in CH-0 CH-1 CH-2 CH-3

G_3 liegt in CH-0 CH-1 CH-2 CH-3

VARIANTEN:

1. $L(G_3) = (a|ba)^*c^+dd$.
2. $L(G_3) = a(ba|ca)^n cdd$ mit $n \geq 0$.
3. $L(G_3) = (ab|ac)^n acdd$ mit $n \geq 0$.
4. $L(G_3) = a((ba)^n(ca)^m)^k cdd$ mit $n \geq 0$.
5. $L(G_3) = ((ab)^n(ac)^m)^k acdd$ mit $n \geq 0$.

$L(G_3)$ liegt in CH-0 CH-1 CH-2 CH-3

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

Die städtische Bibliothek Boxingen speichert ihre Stammdaten in folgenden Tabellen:

Exemplare

Feld	Typ
exNr	int
buchNr	int
einkauf	date

Bücher

Feld	Typ
buchNr	int
titel	String
autor	String

Kunden

Feld	Typ
kundenNr	int
name	String
adresse	String

Ausleihen

Feld	Typ
kundenNr	int
exNr	int
faellig	date

a) Mahngebühren sind die wichtigste Einnahmequelle der Bibliothek. Formulieren Sie eine SQL-Anfrage, welche die Namen aller Benutzer mit überfälligen Ausleihen ermittelt.

VARIANTEN:

- ```
SELECT DISTINCT name
FROM Ausleihen INNER JOIN Kunden ON
 Ausleihen.kundenNr = Kunden.kundenNr
WHERE Ausleihen.faellig < X
```
- ```
SELECT DISTINCT name
FROM   Ausleihen NATURAL JOIN Kunden
WHERE  Ausleihen.faellig < X
```
- ```
SELECT DISTINCT Kunden.name
FROM Kunden, Ausleihen
WHERE Kunden.kundenNr = Ausleihen.kundenNr AND
 Ausleihen.faellig < X
```

HINWEIS:

Statt DISTINCT kann auch der Zusatz GROUP BY name verwendet werden.

**A3**

K1

K2

K3

b) Ein Kunde möchte ein Buch mit gegebener Buchnummer entleihen. Formulieren Sie eine SQL-Anfrage, welche die Nummern der nicht verliehenen Exemplare dieses Buches ermittelt.

VARIANTEN:

1. Left Join:

```
SELECT Exemplare.exNr
FROM Exemplare LEFT JOIN Ausleihen ON
 Exemplare.exNr = Ausleihen.exNr
WHERE buchNr = X AND
 kundenNr IS NULL
```

2. Subselect:

```
SELECT exNr
FROM Exemplare
WHERE buchNr = X AND
 exNr NOT IN (
 SELECT exNr
 FROM Ausleihen
)
```

3. Mengenoperation:

```
SELECT exNr
FROM Exemplare
WHERE buchNr = X
EXCEPT (
 SELECT exNr
 FROM Ausleihen
)
```

HINWEISE:

Typisch war folgender Fehler, der aufgrund des kartesischen Produktes alle Exemplare auswählt, wenn mehr als zwei Ausleihen vorliegen:

```
SELECT Exemplare.exNr
FROM Exemplare, Ausleihen
WHERE Exemplare.buchNr = X AND
 Exemplare.exNr <> Ausleihen.exNr
```

**A3**

K1

K2

K3

c) Unlängst wurde ein Buch der Bibliothek in beschädigtem Zustand aufgefunden. Die Buchnummer war unkenntlich, allein die Exemplarnummer konnte abgelesen werden. Formulieren Sie eine SQL-Anfrage, welche die Exemplarnummern aller Exemplare dieses Buches ermittelt.

VARIANTEN:

1. Self Join:

```
SELECT Exemplare.exNr
FROM Exemplare AS Orig, Exemplare
WHERE Orig.exNr = X AND
 Orig.buchNr = Exemplare.buchNr
```

2. Subselect:

```
SELECT exNr
FROM Exemplare
WHERE buchNr IN (
 SELECT buchNr
 FROM Exemplare
 WHERE exNr = X
)
```

**A3**

K1

K2

K3

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

a) Sei  $\mathcal{B}$  eine boolesche Algebra. Zeigen Sie: Gilt für zwei gegebene  $a, b \in \mathcal{B}$  die Identität  $a \wedge b = a$ , so gilt für sie auch  $a \vee b = b$ . (Hinweis: Beginnen Sie Ihre Umformungen bei  $b$ ).

VARIANTEN:

1. Verschmelzungsgesetz

$$a \vee b \stackrel{\text{Prämisse}}{=} (a \wedge b) \vee b \stackrel{\text{Verschmelzungsgesetz}}{=} b$$

2. (a) Halbordnung

$$a \wedge b = a \Leftrightarrow a \leq b \Leftrightarrow a \vee b = b$$

(b) Dualitätsprinzip

$$a \wedge b = a \Leftrightarrow \inf(a, b) = a \Leftrightarrow \sup(a, b) = b \Leftrightarrow a \vee b = b$$

3. b-Varianten (direkt)

$$(a) \quad b = b \wedge (a \vee \top) \stackrel{\text{Dist.}}{=} (b \wedge a) \vee (b \wedge \top) \stackrel{\text{Prämisse}}{=} a \vee (b \wedge \top) = a \vee b$$

$$(b) \quad b = b \vee (a \wedge \perp) \stackrel{\text{Dist.}}{=} (b \vee a) \wedge (b \vee \perp) = (b \vee a) \wedge b \stackrel{\text{Dist.}}{=} (b \wedge b) \vee (a \wedge b) \stackrel{\text{Prämisse}}{=} b \vee a$$

$$(c) \quad b = b \wedge (a \vee \complement a) \stackrel{\text{Dist.}}{=} (b \wedge a) \vee (b \wedge \complement a) \stackrel{\text{Prämisse}}{=} a \vee (b \wedge \complement a) \stackrel{\text{Dist.}}{=} (a \vee b) \wedge (a \vee \complement a) = a \vee b$$

(d)

$$\begin{aligned} b &= b \vee \perp \\ &= b \vee (a \wedge \complement a) \\ &= (b \vee a) \wedge (b \vee \complement a) \\ &= (a \vee b) \wedge (\complement a \vee b) \end{aligned}$$

Noch zu zeigen:  $\complement a \vee b = \top$ .

$$\begin{aligned} \complement a \vee b &= (\complement a \vee b) \wedge \top \\ &= (\complement a \vee b) \wedge (a \vee \complement a) \\ &= (\complement a \vee b) \wedge (\complement a \vee a) \\ &= \complement a \vee (b \wedge a) \quad (\text{Prämisse}) \\ &= \complement a \vee a = \top \end{aligned}$$

4. Syntaktische und semantische Gleichheit: wandle semantisches Gleichheitszeichen "=" in syntaktische Äquivalenz " $\Leftrightarrow$ " um und beweise in der Algebra:

$$\begin{aligned} &a \vee b = b \\ \Leftrightarrow &a \vee b \leftrightarrow b \\ \Leftrightarrow &(a \vee b) \wedge b \vee \complement(a \vee b) \wedge \complement b \\ \Leftrightarrow &a \wedge b \vee b \wedge b \vee \complement a \wedge \complement b \wedge \complement b \\ \Leftrightarrow &a \wedge b \vee b \vee \complement a \wedge \complement b \\ \Leftrightarrow &a \vee b \vee \complement(a \vee b) \\ \Leftrightarrow &\top \end{aligned}$$

**A4**

K1

K2

K3

## HINWEISE:

1. Folgender Ansatz ist unzureichend, denn Syntax macht noch keinen Beweis:

$a \wedge b = a \Leftrightarrow a \vee b = b$  Links: Prämisse, Rechts: noch zu Beweisen (z.B. mit der Variante Verschmelzungsgesetz)

$a = a \Leftrightarrow b = b$

$true \Leftrightarrow true$

$true$

2. Wertetabellen und  $b = \top, b = a, b = \perp$ : Eine boolesche Algebra kann mehr Elemente als  $\top$  oder  $\perp$  enthalten. Beweise mit Wertetabelle oder Fallunterscheidung sind daher unmöglich.
3. Für alle Elemente  $A, B$  und binären Operatoren " $\odot$ " aus einer beliebigen Algebra gilt, nach Def. der Gleichheit:

$$A = B \implies A \odot X = B \odot X$$

Aber die Umkehrung gilt im Allgemeinen nicht! Insbesondere für die boolesche Algebra ist die Umkehrung nicht richtig.

b) Bringen Sie die aussagenlogische Formel  $F = \neg((x \rightarrow y) \equiv z)$  in konjunktive Normalform.

VARIANTEN:

1. Ausrechnen

$$\begin{aligned}
 F &= \neg((x \rightarrow y) \equiv z) \\
 &= \neg((\bar{x} \vee y) \equiv z) \\
 &= \neg(((\bar{x} \vee y) \wedge z) \vee (\neg(\bar{x} \vee y) \wedge \bar{z})) \\
 &= \neg((\bar{x} \wedge z) \vee (y \wedge z) \vee (x \wedge \bar{y} \wedge \bar{z})) \\
 &= \neg(\bar{x} \wedge z) \wedge \neg(y \wedge z) \wedge \neg(x \wedge \bar{y} \wedge \bar{z}) \\
 &= (x \vee \bar{z}) \wedge (\bar{y} \vee \bar{z}) \wedge (\bar{x} \vee y \vee z)
 \end{aligned}$$

2. Wertetabelle

| x | y | z | F |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 |

Hieraus muß noch die KNF abgeleitet werden.

HINWEISE:

1. Die KNF (konjunktive Normalform) muß nicht minimal sein. Die vollständige KNF lautet:

$$(\neg x \vee \neg y \vee \neg z) \wedge (\neg x \vee y \vee z) \wedge (x \vee \neg y \vee \neg z) \wedge (x \vee y \vee \neg z)$$

**Achtung:** Durch Hinzufügen von Tautologien wie  $(x \vee \neg x)$  können beliebige andere KNFs gewonnen werden (unendlich viele Varianten).

2. Da die DNF leichter erstellbar war, wurde sie oft fälschlicherweise als Lösung angegeben:

$$(\neg x \wedge \neg z) \vee (y \wedge \neg z) \vee (x \wedge \neg y \wedge z)$$

Die vollständige DNF:

$$(\neg x \wedge \neg y \wedge \neg z) \vee (\neg x \wedge y \wedge \neg z) \vee (x \wedge \neg y \wedge z) \vee (x \wedge y \wedge \neg z)$$

**A4**

K1

K2

K3

## Aufgabe 5: Haskell (3+4+4=11 Punkte)

a) Sie müssen dringend eine Zeichenkette umkehren ("hallo" → "ollah"), haben aber versehentlich die Standardbibliothek gelöscht. Nur Mustererkennung und die Listenoperation `:` funktionieren noch. Implementieren Sie mit diesen Mitteln die Funktion `reverse :: [a] -> [a]`.

VARIANTEN:

1. Lösung mit spezieller Hilfsfunktion:

```
reverse xs = reverse' xs []
reverse' [] ys = ys
reverse' (x:xs) ys = reverse' xs (x:ys)
```

2. Lösung mit Redefinition von `++`:

```
reverse [] = []
reverse (x:xs) = (reverse xs) ++ [x]
(++) [] ys = ys
(++) (x:xs) ys = x:(xs ++ ys)
```

3. Lösung mit Redefinition von `last` und `init`:

```
reverse [] = []
reverse xs = (last xs):(reverse (init xs))
last [x] = x
last (x:xs) = (last xs)
init [x] = []
init (x:xs) = x:(init xs)
```

HINWEISE:

1. Nicht korrekt, weil der Operator `:` vorne an Listen anhängt:

```
reverse [] = []
reverse (x:xs) = (reverse xs):x
```

2. Wer `++` benutzt, muß es definieren!

b) Gegeben sei eine duplikatfreie Liste  $l :: [a]$ . Wir bezeichnen die Liste aller möglichen Teillisten mit Auslassungen als Kombinationen  $c :: [[a]]$  von  $l$ . Beispiel: die Kombinationen von  $[1,2,3]$  sind  $[[1,2,3], [1,2], [1,3], [1], [2,3], [2], [3], []]$ . Die relative Position jeder einzelnen Kombination ist dabei unerheblich.

Schreiben Sie eine Funktion  $\text{combs} :: [a] \rightarrow [[a]]$ , welche die Kombinationen ihres Arguments berechnet. Sie können beliebige Funktionen der Standardbibliothek verwenden.

VARIANTEN:

1. (a) 

```
combs [] = [[]]
combs (x:xs) = (map (x:) co) ++ co
 where co = combs xs
```

(b) Gleicher Ansatz in Mengenschreibweise:

```
combs [] = [[]]
combs (x:xs) = [x:ys | ys <- zs] ++ zs
 where zs = combs xs
```

2. Schrittweises Auslassen (erzeugt Doubletten)

```
combs xs = xs:[zs | ys <- auslassungen xs, zs <- combs ys]
auslassungen [] = []
auslassungen (x:xs) = xs:[(x:ys) | ys <- (auslassungen xs)]
```

3. Aufbau über die Potenzmenge

```
combs xs = [comb i xs | i <- [0..max]]
 where max=2^(length xs)-1
```

```
comb _ [] = []
comb i (x:xs) | r==0 = comb q xs
 | r==1 = x : (comb q xs)
 where (q,r) = divMod i 2
```

c) Vereinbaren Sie einen Datentyp `BSTree k v` für binäre Suchbäume. Knoten, die nicht dem leeren Baum entsprechen, tragen Schlüssel vom Typ `k` und Werte vom Typ `v`. Definieren Sie eine Funktion  $\text{contains} :: \text{Ord } k \Rightarrow (\text{BSTree } k \ v) \rightarrow k \rightarrow \text{Bool}$ , die prüft, ob ein binärer Suchbaum einen gegebenen Schlüssel enthält.

```
data BSTree k v = BSEmpty |
 BSInner (BSTree k v) k v (BSTree k v)

contains BSEmpty _ = False
contains (BSInner l k _ r) k' | k < k' = contains l k'
 | k == k' = True
 | k > k' = contains r k'
```

A5

K1

K2

K3

## Aufgabe 6: Asymptotischer Aufwand (3+2+2+3=10 Punkte)

Gegeben sei folgende Funktion:

```
func xs k | k < len = func smaller k
 | k == len = pivot
 | k > len = func (tail larger) (k-len-1)
 where pivot = head xs
 smaller = filter (< pivot) xs
 larger = filter (>=pivot) xs
 len = length smaller
```

a) Geben Sie die Signatur der Funktion an. Beschreiben Sie in einem Satz, was die Funktion tut.

```
func :: Ord a => [a] -> Int -> a
```

Die Funktion gibt das  $k$ t-kleinste Element der Liste  $xs$  zurück.

b) Betrachten Sie zunächst den lokalen Aufwand eines Aufrufes von `func` ohne rekursiven Abstieg. Definieren Sie ein Maß der Problemgröße  $n$  und geben Sie den asymptotischen Aufwand folgender Teilberechnungen an (Operationen auf Listenelementen haben konstanten Aufwand):

```
n := Länge von xs
pivot ist $O(1)$
smaller,
larger sind $O(n)$
len ist $O(n)$
```

c) Nehmen Sie an, die Anordnung der Elemente von  $xs$  sei rein zufällig. Geben Sie die Rekurrenzgleichung für den erwarteten Aufwand  $f$  von `func` an.

$$f(n) = 1 * f(n/2) + O(n)$$

HINWEIS:

Typischer Fehler:  $n - 1$  statt  $n/2$ .

d) Lösen Sie die Rekurrenz. Welchen asymptotischen erwarteten Aufwand hat `func`?

$1 < 2 \Rightarrow f(n) \in O(n)$  (Hauptsatz über Rekurrenzen)

**A6**

K1

K2

K3