

# Probeklausur Informatik I

## 10.01.2003

Vorname:

Name:

Zur Probeklausur sind keine Hilfsmittel zugelassen. Die Bearbeitungszeit beträgt 60 Minuten.

Bitte tragen Sie zunächst Ihren Vornamen und Namen sowie auf jeder Seite Ihre Matrikelnummer und Ihr Tutorium ein.

Die Probeklausur ist komplett und geheftet abzugeben. Sie dürfen die Rückseite der Aufgabenblätter als Konzeptpapier benutzen. Verwenden Sie kein eigenes Papier.

Die errungene Punktezahl und Note dienen allein der Selbstkontrolle. Sie gehen nicht in die Bewertung Ihrer Übungsblätter ein.

Art, Anzahl und Schwierigkeitsgrad der Aufgaben in der Probeklausur sind fiktiv. Jegliche Zusammenhänge mit der eigentlichen Klausur sind unbeabsichtigt.

Aufgabe	1	2	3	4	5	6	7	8	$\Sigma$
Maximal	9	15	6	6	7	6	5	6	60
1. Korr.									
2. Korr.									
3. Korr.									

Punkte:

Note :

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

a) Multiple Choice

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

1. Die Kelleroperation pop ist

*W*  *F* Ein Konstruktor

*W*  *F* Ein Hilfskonstruktor

*W*  *F* Ein Projektor

2. Ein Akzeptor ist ein spezieller

*W*  *F* Moore-Automat

*W*  *F* Mealy-Automat

3. Der gerichtete Graph  $G = (E, K)$  mit  $E = \{1, 2, 3, 4\}$ ,  $K = \{(1, 2), (2, 4), (4, 1), (1, 3), (4, 3)\}$

*W*  *F* ist transitiv

*W*  *F* ist reflexiv

*W*  *F* besitzt einen Hamiltonschen Kreis

*W*  *F* besitzt einen Eulerschen Kreis

4.  *W*  *F* Die Anzahl aller Marken in einem Petrinetz ist konstant.

5.  *W*  *F* Faule und eifrige Auswertung liefern identische Ergebnisse.

6.  *W*  *F* Jeder Regelkreis beinhaltet eine Steuerung.

b) Freitext (1 Punkt pro Antwort)

1. Bilden Sie das Zweierkomplement von 0001011.

2. Definieren Sie "Wissen" im Sinne der Vorlesung.

3. Beschreiben Sie den Flaschenhals des von-Neumann-Rechners.

1. Korr 2. Korr 3. Korr.

## Aufgabe 2: Haskell (6+9=15 Punkte)

Gegeben sei eine Binärdarstellung natürlicher Zahlen als Liste von Booleans, wobei das höchstwertige Bit zuerst steht (d.h., [True, True, False, True] entspricht 13).

a) Schreiben Sie Funktionen, welche Zahlen zwischen unserer Darstellung und dem Haskell-Datentyp `Integer` konvertieren und folgenden Signaturen genügen:

```
binaryToInt :: [Bool] -> Int
intToBinary :: Int    -> [Bool]
```

Verwenden Sie dabei keine vordefinierten Funktionen außer den elementaren arithmetischen und logischen Operatoren. Wenn Sie eigene Hilfsfunktionen schreiben, geben Sie deren Signatur an.

```
binaryToInt x = btiHelper x 0
btiHelper :: [Bool] -> Int -> Int
btiHelper []      acc = acc
btiHelper (True :xs) acc = btiHelper xs (1+2*acc)
btiHelper (False:xs) acc = btiHelper xs ( 2*acc)

intToBinary x = itbHelper x []
itbHelper :: Int -> [Bool] -> [Bool]
itbHelper 0 bs = bs
itbHelper x bs | (mod x 2) == 1 = itbHelper (div x 2) (True :bs)
itbHelper x bs | otherwise      = itbHelper (div x 2) (False:bs)
```

b) Schreiben Sie eine Funktion `binaryAdd :: [Bool] -> [Bool] -> [Bool]`, die zwei Zahlen in unserem Binärsystem addiert (Konvertierung ist unzulässig). Sie dürfen Bibliotheksfunktionen nutzen. Wenn Sie eigene Hilfsfunktionen schreiben, geben Sie deren Signatur an.

```
binaryAdd a b = reverseAdd (fixLen (a,b) ((length a)-(length b)))

fixLen :: ([Bool],[Bool]) -> Int -> ([Bool],[Bool])
fixLen (a,b) delta | delta > 0 = fixLen (a,False:b) (delta-1)
                  | delta < 0 = fixLen (False:a,b) (delta+1)
                  | otherwise = (a,b)

reverseAdd :: ([Bool], [Bool]) -> [Bool]
reverseAdd (a,b) = reverse (baHelper (reverse a) (reverse b) False)

baHelper :: [Bool] -> [Bool] -> Bool -> [Bool]
baHelper [] [] True  = True : []
baHelper [] [] False = []
baHelper (x:xs) (y:ys) c
  = (xor x (xor y c)) : (baHelper xs ys (atLeastTwo x y c))

atLeastTwo :: Bool -> Bool -> Bool -> Bool
atLeastTwo x y c = (x && y) || (x && c) || (y && c)

xor :: Bool -> Bool -> Bool
xor  x y    = (x || y) && (not (x && y))
```

1. Korr 2. Korr 3. Korr.

<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
--------------------------	--------------------------	--------------------------

**Aufgabe 3: SQL (3+3=6 Punkte)**

Gegeben seien folgende Tabellen:

**PersonSport**

Person	Sport
Peter	Fußball
Claudia	Volleyball
Heinz	Hockey
Timo	Volleyball
Paul	Fußball

**PersonStadtteil**

Person	Stadtteil
Peter	Weststadt
Heinz	Oststadt
Paul	Südstadt
Timo	Oststadt
Claudia	Weststadt

**StadtteilSport**

Stadtteil	Sport
Weststadt	Volleyball
Weststadt	Hockey
Weststadt	Basketball
Oststadt	Fußball
Südstadt	Hockey

a) Formulieren Sie folgende Abfrage einmal mit einem Join und einmal als geschachtelte Abfrage in SQL: "Welche Stadtteile bieten mindestens ein Sportangebot für Person X?"

```
SELECT UNIQUE StadtteilSport.Stadtteil
FROM StadtteilSport, PersonSport
WHERE PersonSport.Person = X AND
      StadtteilSport.Sport = PersonSport.Sport

SELECT UNIQUE Stadtteil
FROM StadtteilSport
WHERE Sport IN (SELECT Sport
                FROM PersonSport
                WHERE Person = X )
```

b) Formulieren Sie folgende Abfrage in SQL: "Welche Personen wohnen in einem Stadtteil, in dem sie mindestens eine ihrer bevorzugten Sportarten ausüben können?"

```
SELECT UNIQUE PersonSport.Person
FROM PersonSport, PersonStadtteil, StadtteilSport
WHERE PersonSport.Sport = StadtteilSport.Sport AND
      PersonSport.Person = PersonStadtteil.Person AND
      PersonStadtteil.Stadtteil = StadtteilSport.Stadtteil
```

1. Korr    2. Korr    3. Korr.

## Aufgabe 4: Abstrakte Datentypen (1+3+2=6 Punkte)

Implementieren Sie den Datentyp Keller von `Int` mit Hilfe des Datentyps `Array` von `Int` mit Schlüssel `Int`. Zur Auffrischung:

```
import Array
type ourArray = Array Int Int           // Datentypdefinition
array (indexVon, indexBis) [(index, wert)] // Erzeugen und Initialisieren
arrayname ! index                       // Lesen
arrayname // [(index, wert)]           // Schreiben
bounds arrayname                         // Grenzen (indexVon, indexBis)
```

a) Wie speichern Sie einen Keller von `Int` in einem `ourArray`? Beschreiben Sie kurz Ihre Kerngedanken.

Verwende den 0. Eintrag des Arrays, um einen Zeiger auf den obersten Kellereintrag zu speichern. Vergrößere das Array, sobald dieser Zeiger die Obergrenze des Arrays erreicht.

b) Implementieren Sie folgende Funktionen in Haskell:

```
createStack :: ourArray
pop          :: ourArray -> ourArray
top         :: ourArray -> Int
isEmpty     :: ourArray -> Bool
```

```
initialSize :: Int
initialSize = 8

createStack = array (0,initialSize) [(0, 0)]
pop arr | (arr !! 0) > 0 = arr // [(0, (arr ! 0) - 1)]
top arr | (arr !! 0) > 0 = arr ! (arr ! 0)
isEmpty arr = (arr ! 0) == 0
```

c) Implementieren Sie die Funktion `push :: ourArray -> Int -> ourArray` in Haskell oder beschreiben Sie Ihre Funktionsweise kurz in Worten.

```
push arr x | ptr < end = arr // [(nxt,x),(0,nxt)]
           | otherwise = array (0,2*end) (assocs arr) // [(nxt,x),(0,nxt)]
           where ptr = arr ! 0
                 nxt = ptr+1
                 end = snd (bounds arr)
```

1. Korr 2. Korr 3. Korr.

<input type="text"/>	<input type="text"/>	<input type="text"/>
----------------------	----------------------	----------------------

**Aufgabe 5: Chomsky Grammatiken / Markov (1+2+4=7 Punkte)**

Sei  $G = (\Sigma, N, P, S)$  mit  $\Sigma = \{a, b, c, d\}$ ,  $N = \{A, B, C, D, E\}$  und den Produktionen

$$\begin{aligned}
 P = \{ & S \rightarrow ABCD, \\
 & CD \rightarrow E, \\
 & E \rightarrow D \mid \epsilon, \\
 & A \rightarrow a \mid aA \\
 & B \rightarrow abc \mid aBbc, \\
 & D \rightarrow d \\
 & C \rightarrow \epsilon \quad \quad \quad \}
 \end{aligned}$$

a) Geben Sie den Chomsky-Typ der Grammatik an. Welche Sprache erzeugt die Grammatik?

$G$  liegt in CH-0.

$L(G) = a^l a^m (bc)^m d^n$ , wobei  $l \geq 1$ ,  $m \geq 1$  und  $n \in \{0, 1\}$ .

b) Definieren Sie eine Grammatik  $G'$  mit  $L(G') = L(G)$ , die in derselben Chomsky-Klasse wie  $L(G)$  liegt. Vermeiden Sie Kettenproduktionen und redundante Produktionen.

$G' = (\Sigma, N', P', S)$  mit  $\Sigma, S$  wie oben,  $N' = \{S, A, B\}$  und den Produktionen

$$\begin{aligned}
 P' = \{ & S \rightarrow AB \mid ABd, \\
 & A \rightarrow a \mid aA \\
 & B \rightarrow abc \mid aBbc \}
 \end{aligned}$$

c) Gegeben sei eine Darstellung der positiven natürlichen Zahlen als Binärzahl über den Ziffern  $\{0, 1\}$ , wobei die höchstwertige Stelle links stehe. Schreiben Sie einen gesteuerten Markov-Algorithmus, der den Vorgänger einer gegebenen Zahl berechnet.

$$\begin{aligned}
 0b &\rightarrow b1 \\
 1b &\rightarrow .0 \\
 a0 &\rightarrow 0a \\
 a1 &\rightarrow 1a \\
 a &\rightarrow b \\
 \epsilon &\rightarrow a
 \end{aligned}$$

1. Korr    2. Korr    3. Korr.

--	--	--

**Aufgabe 6: Boolesche Algebra (2+2+2=6 Punkte)**

a) Zeigen Sie: “ $\mathcal{C}a \rightarrow ((c \vee \mathcal{C}d) \rightarrow \mathcal{C}(b \wedge a))$  ist eine Tautologie”.

$$\begin{aligned}
 & \mathcal{C}a \rightarrow ((c \vee \mathcal{C}d) \rightarrow \mathcal{C}(b \wedge a)) \\
 \equiv & \mathcal{C}\mathcal{C}a \vee ((c \vee \mathcal{C}d) \rightarrow \mathcal{C}(b \wedge a)) \\
 \equiv & a \vee (\mathcal{C}(c \vee \mathcal{C}d) \vee \mathcal{C}(b \wedge a)) \\
 \equiv & a \vee ((\mathcal{C}c \wedge d) \vee (\mathcal{C}b \vee \mathcal{C}a)) \\
 \equiv & a \vee \mathcal{C}a \vee \mathcal{C}b \vee (\mathcal{C}c \wedge d) \\
 \equiv & \top \vee \mathcal{C}b \vee (\mathcal{C}c \wedge d) \\
 \equiv & \top
 \end{aligned}$$

b) Leiten Sie das Verschmelzungsgesetz  $x \vee (x \wedge y) \equiv x$  aus anderen Gesetzen der booleschen Algebra her. Verwenden Sie dabei nicht  $x \wedge (x \vee y) \equiv x$ .

$$\begin{aligned}
 & x \vee (x \wedge y) \equiv x \\
 \equiv & (x \wedge \top) \vee (x \wedge y) \equiv x \\
 \equiv & x \wedge (\top \vee y) \\
 \equiv & x \wedge \top \\
 \equiv & x
 \end{aligned}$$

c) Beweisen oder widerlegen Sie folgende Aussage:

“Wenn  $a \rightarrow b$  erfüllbar und  $b$  keine Tautologie ist, so ist  $a$  erfüllbar”.

Gegenbeispiel: Setze  $a = x \wedge \mathcal{C}x$ , so ist  $a \equiv \perp$ . Damit ist  $a \rightarrow b \equiv \top$  unabhängig von  $b$ , insbesondere also für  $b = x \vee \mathcal{C}x$ .

1. Korr	2. Korr	3. Korr.
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

### Aufgabe 7: Prädikatenlogik (2+3=5 Punkte)

Bringen Sie folgende Formeln zunächst in bereinigte Skolemform. Stellen Sie dafür zuerst die bereinigte Pränexform her.

a)  $\forall x : (F(x) \wedge \exists x : (B(x) \rightarrow C(x)))$

$$\begin{aligned} & \forall x : (F(x) \wedge \exists x : (B(x) \rightarrow C(x))) \\ \equiv & \forall x : (F(x) \wedge \exists y : (B(y) \rightarrow C(y))) \\ \equiv & \forall x : (F(x) \wedge \exists y : (\neg B(y) \vee C(y))) \\ \equiv & \forall x \exists y : (F(x) \wedge (\neg B(y) \vee C(y))) \\ \equiv & \forall x : (F(x) \wedge (\neg B(sk_1(x)) \vee C(sk_1(x)))) \end{aligned}$$

b)  $((\exists x \exists y : P(x, y)) \rightarrow (\forall x \forall y : P(x, y))) \wedge \exists y : P(y, y)$

$$\begin{aligned} & ((\exists x \exists y : P(x, y)) \rightarrow (\forall x \forall y : P(x, y))) \wedge \exists y : P(y, y) \\ \equiv & ((\exists x \exists y : P(x, y)) \rightarrow (\forall z \forall w : P(z, w))) \wedge \exists u : P(u, u) \\ \equiv & (\neg(\exists x \exists y : P(x, y)) \vee (\forall z \forall w : P(z, w))) \wedge \exists u : P(u, u) \\ \equiv & ((\forall x \forall y : \neg P(x, y)) \vee (\forall z \forall w : P(z, w))) \wedge \exists u : P(u, u) \\ \equiv & \forall x \forall y \forall z \forall w \exists u : ((\neg P(x, y) \vee P(z, w)) \wedge P(u, u)) \\ \equiv & \forall x \forall y \forall z \forall w : ((\neg P(x, y) \vee P(z, w)) \wedge P(sk_1(x, y, z, w), sk_1(x, y, z, w))) \end{aligned}$$

1. Korr    2. Korr    3. Korr.

<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
--------------------------	--------------------------	--------------------------

## Aufgabe 8: Lambda-Kalkül (4+2=6 Punkte)

a) Gegeben sei folgende Funktion zur Berechnung von Fakultäten:

$$FAK = \lambda a.(IF(= a 0) 1 (* a (FAK(- a 1))))$$

Werten Sie den Funktionsaufruf mit 2 aus. Dabei dürfen Sie FAK, IF, TRUE, FALSE, \*, -, = und die Zahlen als Kurzform und nicht im Lambda-Kalkül benutzen. Lassen Sie keine Zwischenschritte aus.

$$\begin{aligned}
 FAK\ 2 &\equiv \lambda a.(IF(= a 0) 1 (* a (FAK(- a 1))))\ 2 \\
 &\equiv^{\beta} IF(= 2 0) 1 (* 2 (FAK(- 2 1))) \\
 &\equiv IF\ FALSE\ 1 (* 2 (FAK(- 2 1))) \\
 &\equiv (* 2 (FAK(- 2 1))) \\
 &\equiv (* 2 (FAK\ 1)) \\
 &\equiv (* 2 (\lambda a.(IF(= a 0) 1 (* a (FAK(- a 1))))\ 1)) \\
 &\equiv^{\beta} (* 2 (IF(= 1 0) 1 (* 1 (FAK(- 1 1)))))) \\
 &\equiv (* 2 (IF\ FALSE\ 1 (* 1 (FAK(- 1 1)))))) \\
 &\equiv (* 2 (* 1 (FAK(- 1 1)))) \\
 &\equiv (* 2 (* 1 (FAK\ 0))) \\
 &\equiv (* 2 (* 1 (\lambda a.(IF(= a 0) 1 (* a (FAK(- a 1))))\ 0))) \\
 &\equiv^{\beta} (* 2 (* 1 (IF(= 0 0) 1 (* 0 (FAK(- 0 1))))\ 0))) \\
 &\equiv (* 2 (* 1 (IF\ TRUE\ 1 (* 0 (FAK(- 0 1))))\ 0))) \\
 &\equiv (* 2 (* 1\ 1)) \\
 &\equiv (* 2\ 1) \\
 &\equiv 2
 \end{aligned}$$

b) Schreiben Sie die Funktion *XOR* ("entweder oder"). Wenn Sie Abkürzungen (z.B. *TRUE*, *IF*...) benutzen möchten, müssen Sie diese vorher definieren ( z.B.  $TRUE = \lambda a.\lambda b.(a)$  ).

$$\begin{aligned}
 TRUE &= \lambda a.\lambda b.(a) \\
 FALSE &= \lambda a.\lambda b.(b) \\
 NOT &= \lambda a.(a\ FALSE\ TRUE) \\
 \\ \\
 XOR &= \lambda x.\lambda y.(x\ (y\ FALSE\ TRUE)\ y)
 \end{aligned}$$

1. Korr    2. Korr    3. Korr.

--	--	--