PROBEKLAUSUR ZU ‘PRAKTISCHE INFORMATIK I’ WS 2001/2002

LEONIE DRESCHLER-FISCHER
UND
WOLFGANG MENZEL

Die Klausur bitte auf jeden Fall geheftet lassen! Kein zusätzliches Papier verwenden!

Name: ______________________________

Matrikelnummer: ______________________________

Hauptfach: ______________________________

Ich bestätige hiermit, daß ich an der Klausur "Praktische Informatik 1 (P1)" zum __________ Mal teilnehme. Falls ich zum dritten Mal an dieser Klausur teilnehme, bestätige ich weiterhin, daß ich an einer Studienberatung nach § 11 Abs. 1 Satz 1 der Prüfungsordnung vom 8.4.98/13.5.98 teilgenommen habe und die entsprechende Bescheinigung

- der Klausur beigelegt habe
- im Prüfungsamt III abgegeben habe
- in der Prüfungsverwaltung des Fachbereichs Informatik abgegeben habe

(Nichtzutreffendes bitte streichen).

Hinweis: Nicht wahrheitsgemäß gegebene Antworten stellen eine Täuschungshandlung dar; nach § 10 Abs. 3 der Prüfungsordnung muß die Klausur in diesem Fall als "nicht bestanden" gewertet werden.

Unterschrift: ______________________________

WICHTIG: Punkte für Teilaufgaben werden durch {⊘} angegeben, z.B. {⊘ ⊘} = 2 Punkte. Die Gesamtanzahl von Punkten pro Aufgabe steht jew. im Kasten am Rand der entsp. Aufgabe unter "von". **Sollte der Platz für eine Aufgabe nicht ausreichen, so verwenden Sie die Leerseiten am Ende und machen Sie einen Vermerk am Rand, etwa "⟹ Seite 15".**

Teil 1: Relationale Programmierung

1. Unifizieren Sie die folgenden Ausdrücke (falls möglich) und geben Sie die dabei erzeugten Variablenbindungen an.

von
6

<table>
<tr><th>Ausdruck 1
Ausdruck 2</th><th>Variablenbindungen</th><th></th></tr>
<tr><td><code>alter(hans,E)</code>
<code>alter(F,klein)</code></td><td></td><td>{⊘ ⊘}</td></tr>
<tr><td><code>t(x,y,y)</code>
<code>t(A,B,A)</code></td><td></td><td>{⊘ ⊘}</td></tr>
<tr><td><code>p([[[m],m],m,[m]],[m])</code>
<code>p([[A|B]|C],A)</code></td><td></td><td>{⊘ ⊘}</td></tr>
</table>

2. Eine relationale Datenbank für Bibliotheksausleihen sei als Faktensammlung in Prolog implementiert und enthält die Relationen `ausleihe(Signatur,Lesernummer)` für jedes derzeit ausgeliehene Buch, `vorbestellung(Signatur,Lesernummer)` für jedes derzeit vorbestellte Buch, und `leser(Name,Vorname,Lesernummer,Adresse,Geburtsjahr)` für die persönlichen Angaben jedes Lesers.
   Stellen Sie die folgenden Anfragen an die Datenbank. Formulieren Sie ihre Anfragen so, dass möglichst wenig überflüssige Information präsentiert wird und die Systemausgabe weitgehend selbsterklärend ist.

von
10

   (a) Welche Lesernummer hat Susi Sorglos? {⊘ ⊘}

(b) Welcher Leser (identifiziert durch Name und Vorname) hat das Buch mit der Signatur `BUG17456` vorbestellt? {⊘ ⊘ ⊘ ⊘}

(c) Welche Bücher, die der Leser mit der Lesernummer 56245 ausgeliehen hat, können verlängert werden (d.h. sind nicht vorbestellt)? Hinweis: Die Negation einer Bedingung kann durch das Prädikat `not/1` erfolgen. {⊘ ⊘ ⊘ ⊘}

3. Gegeben seien die beiden folgenden Prädikatsdefinitionen:

```
p(X,Y) :- pp(X,[ ],Y).

pp([ ],X,X).
pp([A|X],Y,Z) :- pp(X,[A|Y],Z).
```

von
13

(a) Was berechnet das Prädikat `p/2`? {⊘ ⊘}

(b) Ersetzen Sie die in der Prädikatsdefinition verwendeten Namen durch selbsterklärende Bezeichner. Vervollständigen Sie die Definition durch einen geeigneten Kommentar! {⊘ ⊘ ⊘}

(c) Mit welchen Aufrufen würden Sie das Prädikat testen? Welche Resultate erwarten Sie jeweils? {⊘ ⊘ ⊘ ⊘ ⊘ ⊘}

(d) Welche Eigenschaften (symmetrisch, reflexiv, transitiv, 1:m, m:1) hat die durch `p/2` definierte Relation? {⊘ ⊘}

Zutreffendes ankreuzen	symmetrisch	reflexiv	transitiv	1:m	m:1
Ja	◯	◯	◯	◯	◯
Nein	◯	◯	◯	◯	◯

4. Wozu dient die funktionale Auswertungsumgebung in Prolog und warum braucht man sie?

von
6

5. Der Bestand an Kraftfahrzeugen einer Firma sei als Liste von zweielementigen Listen gegeben, wobei das erste Element einer Unterliste das polizeiliche Kennzeichen und das zweite das Baujahr angibt, z.B.

von
15

```
[[hh-gu_12-67, 1992],
 [hh-wa_34-25, 1987],
 [hh-ig_84-62, 1998],
  ...
 [hh-ba_39-29, 1983]]
```

Die Leitung des Unternehmens benötigt einen Überblick über die Altersstruktur der vorhandenen Fahrzeuge und bittet Sie um verschiedene Informationen. Definieren Sie geeignete Prädikate um den jeweiligen Informationsbedarf zu befriedigen und geben Sie die erforderlichen Prädikatsaufrufe an.

(a) Wieviele Kraftfahrzeuge sind derzeit im Bestand? $\{\oslash\oslash\oslash\}$

(b) Welche Fahrzeuge sind älter als eine vorgegebene (frei wählbare) Anzahl von Jahren?
Hinweis: Erzeugen Sie eine Liste der betreffenden polizeilichen Kennzeichen. $\{\oslash\oslash\oslash\oslash\oslash\oslash\}$

(d) Was ist das Baujahr des ältesten Fahrzeugs? $\{\oslash \oslash \oslash \oslash \oslash \oslash\}$

Teil 2: Funktionale Programmierung in Scheme[1]

von
15

1. Zu welchen Werten evaluieren die folgenden Scheme-Ausdrücke?

   (a) `(+ 3 (- 6 7))`

   (b) `'(+ 3 (- 6 7))`

   (c) `(car '(auto bus))`

   (d) `(cdr '(auto bus))`

   (e) `(car '((auto bus)))`

   (f) `(map sqrt '(1 9 4 25))`

   (g) `(map number? '(2 auto bus (1 2)))`

   (h) `(map + '(1 2 3) '(2 3 4))`

   (i) `(map (lambda (x) (* x x x)) '(1 2 3))`

   (j) `(reduce * '(1 2 6 0 1) 1)`

   (k) `((curry < 6) 2)`

[1] Beachten Sie das Funktionslexikon auf der letzten Seite!

2. Geben Sie einen Scheme-Ausdruck an, für den den die Auswertung durch innere Reduktion vorteilhafter ist als die äußere Reduktion. Begründung?

von
3

3. Ausdrücke mit `if` und `cond` werden in Scheme anders ausgewertet als funktionale Ausdrücke. Worin besteht der Unterschied, und warum ist das notwendig?
   Nennen Sie zwei weitere *special form operators*.

von
5

von
9

4. Gegeben seien die folgenden Funktionsdefinitionen:

```
(define (alle-neune xs)
   (cond ((null? xs) 0)
         ((= 9 (car xs)) (+ 1 (alle-neune (cdr xs))))
         (else (alle-neune (cdr xs)))))

(define (alle-zehne  xs wieviele)
   (cond ((null? xs) wieviele)
         ((= 10 (car xs)) (alle-zehne (cdr xs) (+ 1 wieviele)))
         (else (alle-zehne (cdr xs) wieviele))))

(define (anzahl-atome xs)
  (cond ((null? xs) 0)
        ((atom? xs) 1)
        (else (+ (anzahl-atome (car xs))
                 (anzahl-atome (cdr xs))))))
```

Betrachten Sie die angegebenen Funktionen `alle-neune, alle-zehne` und `anzahl-atome`. Geben Sie für jede dieser Funktionen an, welche Art von Rekursion vorliegt (lineare Rekursion, Baumrekursion, Endrekursion) und warum.

5. Funktionen höherer Ordnung: Gegeben sei eine Liste von Wertpaaren `(`$(x_1.y_1)$`,`$(x_2.y_2)$`, ...`$(x_n.y_n)$`)`. Definieren Sie Scheme-Funktionen zur Berechnung der folgenden Werte. Verwenden Sie dabei nach Möglichkeit Funktionen höherer Ordnung, wie `map, reduce, curry` usw.

von
18

(a) Eine Funktion `xliste`, die aus einer Liste von Wertpaaren die Liste aller x-Komponenten extrahiert, z.B. `(xliste '((1 . 3) (2 . 4) (3 . 5)))` evaluiert zu `(1 2 3)`. $\{\oslash\,\oslash\}$

(b) Eine Funktion `yliste`, die aus einer Liste von Wertpaaren die Liste aller y-Komponenten extrahiert, z.B. `(yliste '((1 . 3) (2 . 4) (3 . 5)))` evaluiert zu `(3 4 5)`. $\{\oslash\,\oslash\}$

(c) Eine Funktion `xsumme`, die die x-Werte einer Liste von Wertpaaren addiert $\sum_{i=1}^{n} x_i$: `(xsumme '((1 . 3) (2 . 4) (3 . 5)))` evaluiert zu `6`. $\{\oslash\,\oslash\,\oslash\}$

(d) Eine Funktion `x*y-Summe`, die die xy-Produkte einer Liste von Wertpaaren addiert $\sum_{i=1}^{n} x_i \cdot y_i$:
`(x*y-Summe '((1 . 3) (2 . 4) (3 . 5)))` evaluiert zu `26`. $\{\oslash \oslash \oslash\}$

(e) Eine Funktion `x**2-Summe`, die die Quadrate der x-Werte einer Liste von Wertpaaren addiert $\sum_{i=1}^{n} x_i^2$:
`(x**2-Summe '((1 . 3) (2 . 4) (3 . 5)))` evaluiert zu `14`. $\{\oslash \oslash \oslash\}$

(f) Die Gerade $y = m \cdot x + c$, deren quadratischer Anpassungsfehler für eine gegebene Menge von Wertpaaren $\{(x_1, y_1), \cdots, (x_n, y_n)\}$ am kleinsten ist, hat die Steigung m und den Achsenabschnitt c mit den folgenden Werten:

$$\begin{aligned} m &= \frac{n \cdot \sum_{i=1}^{n}(x_i \cdot y_i) - \sum_{i=1}^{n} x_i \cdot \sum_{i=1}^{n} y_i}{n \cdot \sum_{i=1}^{n} x^2 - \sum_{i=1}^{n} x_i \cdot \sum_{i=1}^{n} x_i} \\ c &= \frac{\sum_{i=1}^{n} y_i - m \cdot \sum_{i=1}^{n} x_i}{n} \end{aligned}$$

Definieren Sie eine Scheme-Funktion höherer Ordnung `anpassungsgerade`, die für eine Menge von Wertpaaren die Anpassungsgerade berechnet und eine Scheme-Funktion als Wert zurückgibt, die die y-Werte der Geraden als Funktion von x berechnet.

```
(define g-anp (anpassungsgerade '((1 . 0.5) (2
. 1) (3 . 1.5))))
```

`(g-anp 2)` evaluiert zu 1, denn die optimale Anpassungsgerade ist die Gerade $y = 0.5 \cdot x$, und der y-Wert an der Stelle $x = 2$ ist 1.
$\{\oslash \oslash \oslash \oslash \oslash\}$

Funktionslexikon

• Verknüpfung von Funktionen

`curry f arg1 ... argn`

Partielle Anwendung von `f` auf `arg1 ... argn` von links nach rechts

`rcurry f argm ... argn`

Partielle Anwendung von `f`, Bindung der Argumente von rechts nach links

`compose f1 ... fn`

Funktionskomposition, Hintereinanderausführung
`(f1 (... (fn x)))`

`conjoin p1? ... pn?`

Konjunktion von Prädikaten

`disjoin p1? ... pn?`

Disjunktion von Prädikaten

`always x`

Die konstante Funktion, deren Wert unabhängig von den Argumenten `x` ist

• Idiome der funktionalen Programmierung

`apply f xs`

Anwendung einer Funktion `f` auf eine Liste von Argumenten `xs`

`map f xs ys zs ...`

Paralles Abbilden einer Liste `xs` oder mehreren Listen auf die Liste der Bilder der Elemente

`filter p? xs`

Die Liste der Elemente von `xs`, die `p?` erfüllen

`reduce f xs seed`

Paarweise Verknüpfung der Elemente von `xs` mit `f`, Startwert `seed`,
`(f x1 (f x2 (... (f seed))))`

`iterate f end? start`

Die Liste der Funktionsanwendungen, bis `end?` erfüllt ist,
`(start (f start) (f (f start)) ...)`

`until f end? start`

Der erste Wert der Folge `start, (f start), (f (f start)), ...` der `end?` erfüllt

Relationale Programmierung	
Aufgabe	Punkte
1	
2	
3	
4	
5	
Summe:	

Funktionale Programmierung	
Aufgabe	Punkte
1	
2	
3	
4	
5	
Summe:	

von
100

Bestanden?	
Ja ◯	Nein ◯

Bestanden bei $\geq$ 50 Punkten		
Punkte	Note	Bestanden?
95 – 100	1	Ja
90 – 94.5	1-	Ja
85 – 89.5	2+	Ja
80 – 84.5	2	Ja
75 – 79.5	2-	Ja
70 – 74.5	3+	Ja
65 – 69.5	3	Ja
60 – 64.5	3-	Ja
55 – 59.5	4+	Ja
50 – 54.5	4	Ja
45 – 49.5	4-	Nein
40 – 44.5	5 +	Nein
35 – 39.5	5	Nein
30 – 34.5	5 -	Nein
0 – 29.5	6	Nein

Datum & Unterschrift KorrektorIn 1 ______________________________

Datum & Unterschrift KorrektorIn 2 ______________________________