Hauptklausur zu Informatik 1
Gedächtnisprotokoll

Klaus Ostermann

16. Februar 2022

Hilfsmittel: Stift
Zeit: 90min
Keine Garantie auf korrekte Aufgaben.
Die Reihenfolge der Aufgaben war je Klausur randomisiert.

1 Programmieraufgaben

Aufgabe 1

Beschreibung: Ein Bauteil („part“) besteht aus einem Namen („name“), einem Herstellungspreis („cost“) und Bestandteilen („components“), die selbst wieder Bauteile sind und Bestandteile haben können. Die Gesamtkosten eines Bauteils sind der Herstellungspreis und die Summe der Preise der Bestandteile.

Teil 1

Programmieren Sie eine Datendefinition für Bauteile („part“) und geben Sie ein Beispiel mit zwei Bestandteilen mit Konstantendefinitionen an.

Teil 2

Programmieren Sie eine Funktion `total-cost`, die ein Bauteil annimt und die Gesamtkosten berechnet.
Tipp: Sie können die Aufgabe entweder mit zwei rekursiven Funktionen oder mithilfe von higher-order-functions (`map`, `foldr`) lösen.

Aufgabe 2

Teil 1

Programmieren Sie eine Funktion `contains-smaller-zero?` nach der Signatur mithilfe von Rekursion.

```
; (list-of Number) -> Boolean
(check-expect (contains-smaller-zero? 1 2 3 -4) #true)
(check-expect (contains-smaller-zero? 1 2 3 0) #false)
```

Teil 2

Programmieren Sie eine higher-order Funktion `contains?` mithilfe von Rekursion und einem Prädikat mit der Signatur `(X -> Boolean)`.
Tipp: Orientieren Sie sich an Ihrer Funktion aus Teil 1.

Teil 3

Programmieren Sie `contains-smaller-zero?` und `contains-empty-string?` mithilfe der `contains?` Funktion aus Teil 2, sodass die folgenden Tests erfüllt werden.

```
(check-expect (contains-empty-string? (list "a" "b" "c" "")) #true)
(check-expect (contains-empty-string? (list "a" "b" "c" "d")) #false)
```

2 Multiple-Choice

Es gab 40 Fragen mit je 4 Antwortmöglichkeiten mit je einer korrekten Antwort.

Frage 1

Was ist der Vorteil des Templates für Summentypen?
Das Template stellt sicher, dass man...

- ☐ alle Alternativen des Summentyps durch Funktionen erzeugen kann.
- ☐ alle Alternativen des Summentyps im Funktionskörper behandelt.
- ☐ alle Alternativen des Summentyps durch Tests abdeckt.
- ☐ alle Alternativen des Summentyps durch Prädikate voneinander unterscheiden kann.

Frage 2

Wozu reduziert der folgende Ausdruck in der BSL in einem Schritt?

```
(cond
  [#true (+2 2)]
  [(> 1 0) (+ 1 1)])
```

- ☐ `(+ 2 2)`
- ☐ `(cond [#true (+ 2 2)])`
- ☐ `(cond [#true 4] [(> 1 0) (+ 1 1)])`
- ☐ `4`

Frage 3

Was ergibt die Auswertung des folgenden Ausdrucks?

```
(quasiquote (+ 4 (unquote (+ 2 5)) 5))
```

- ☐ `(list + 4 (list + 2 5))`
- ☐ `(list '+ 4 7 5)`
- ☐ `` `(+ 4 ,(+ 2 5) 5) ``
- ☐ `(list + 4 7 5)`

Frage 4

Was ist eine wichtige Eigenschaft von Tests?

- ☐ Wenn alle Tests wie erwartet ablaufen, ist das Programm fehlerfrei.
- ☐ Tests können als Teil der Spezifikation einer Funktion verwendet werden.
- ☐ Tests stellen sicher, dass eine Rekursion immer terminiert.
- ☐ Mit Tests kann garantiert werden, dass eine Funktion im Programm immer mit den richtigen Typen aufgerufen wird.

Gegeben sei folgendes Programm:

```
(define (f x)
  (cond
    [(empty? x) empty]
    [(cons? x) (cons (first x) (g (rest x)))]))

(define (g x)
  (cond
    [(empty? x) empty]
    [(empty? (rest x)) empty]
    [(cons? x) (f (rest (rest x)))]))
```

Frage 5

Zu was wird folgender Ausdruck ausgewertet?

```
(f (list #true 24 48 72 #false))
```

- ☐ `(list #true 24 48 72 #false)`
- ☐ `(list #true 72)`
- ☐ `(list (list #true) (list 48 #false))`
- ☐ `(list #true 48 #false)`

Frage 6

Was ist die Signatur von `f`?

- ☐ `(List-Of Number) -> (List-Of Number)`
- ☐ `Number -> Boolean`
- ☐ `[X] (List-Of X) -> (List-Of X)`

Frage 7

Was ist die Signatur der folgenden Funktion?

```
(define (composition f g) (lambda (x) (g (f x))))
```

- ☐ `[X Y Z] (X -> Y) (Y -> Z) -> (X -> Z)`
- ☐ `[X Y Z] (Y -> Z) (X -> Y) -> (X -> Z)`

☐ `[X Y Z] (X -> Z) (Z -> Y) -> (Y -> Z)`

Gegeben sei folgendes Programm:

```
(define-struct collision (left right top bottom))
; Collision is a structure: (make-collision Boolean Boolean Boolean Boolean)
```

Frage 8

Wie viele Werte umfasst der Teil des Datenuniversums, der von `collision` *beschrieben wird?*

- ☐ unendlich viele
- ☐ 1
- ☐ 4
- ☐ 16

Frage 9

Was ist eine `collision`*?*

- ☐ rekursiver Datentyp
- ☐ Summentyp
- ☐ Produkttyp
- ☐ Intervalltyp

Frage 10

Was ist eine gültige collision?

- ☐ `<make-collision #t 42 #t "hello">`
- ☐ `<make-collision #t #f #t #f #t>`
- ☐ `<make-collision #t #t <make-collision #t #f>>`
- ☐ `<make-collision #t #t #t #f>`

Frage 11

Was beschreibt die erste Zeile des obigen Programms?

```
(define-struct collision (left right top bottom))
```

- ☐ Strukturdefinition
- ☐ Produkttyp
- ☐ Strukturtyp
- ☐ Keine der Antworten

Frage 12

Sie möchten ein Programm schreiben, das ein textbasiertes Rollenspiel implementiert. Dazu möchten Sie die Charakterklassen „Mage“, „Warrior“ und „Amazon“ einbeziehen. Welchen Datentyp wählen Sie dafür?

- ☐ Summentyp
- ☐ Produkttyp
- ☐ Intervalltyp
- ☐ Primitiver Datentyp

Frage 13

Sie möchten einen Charakter in einem Computerspiel mit einem Namen, einem Level und Werten für Stärke, Intelligenz, Charisma und Geschicklichkeit in einem Programm repräsentieren. Welche Art Datentyp sollten Sie hierzu verwenden?

- ☐ Summentyp
- ☐ Produkttyp
- ☐ Rekursiver Datentyp
- ☐ Primitiver Datentyp

Frage 14

Welches der folgenden Programme ist eine Umgebung?

- ☐
```
(define (f x) (+ x 1))
(define y (f 2))
```
- ☐
```
(define (f x) (+ x 1))
(define y x)
```
- ☐
```
(define (f x) (+ x 1))
(define y 3)
```
- ☐
```
(define (f x) (+ x 1))
(define y (+ 2 1))
```

Frage 15

Zu was wird folgender Ausdruck ausgewertet?

```
(+ 1 "1")
```

- ☐ Fehler
- ☐ `11`
- ☐ `"11"`
- ☐ `2`

Frage 16

Wann kommen die ersten Tests gemäß des Entwurfsrezepts?

- ☐ Nach den Templates
- ☐ Nach der Implementierung
- ☐ Nach den Datenbeispielen
- ☐ Nach dem Funktionskopf

Frage 17

Was ergibt die Auswertung des folgenden Ausdrucks?

```
(#true + 3)
```

- ☐ Fehler: erwartet Zahl als erstes Argument
- ☐ `#true`
- ☐ Fehler: erwartet Funktion nach öffnender Klammer
- ☐ `4`

Frage 18

Gegeben sei folgende kontextfreie Grammatik für E-Mail Adressen.

```
<NAME>  ::= maria | elke | doris | maier | <NAME>-<NAME>
<TLD>   ::= de | com | co.uk
<URL>   ::= amazon | google | tuebingen | <URL>.<URL>
<EMAIL> ::= <NAME>@<URL>.<TLD>
```

*Was ist **keine** korrekte E-Mail Adresse im Sinne der gegebenen Grammatik?*

- ☐ elke@tuebingen.amazon.de
- ☐ maria-maier@google.com
- ☐ maier@google-tuebingen.com
- ☐ doris-maria@google.co.uk

Gegeben sei folgendes Programm:

```
(define (myfun x)
  (cond
    [(> 0 x) (+ x 1)]
    [else #false]))
```

Frage 19

Was ist die Signatur von myfun?

- ☐ `Boolean -> Number`
- ☐ `MaybeNumber -> Number`
- ☐ `Number -> MaybeNumber`

☐ `Number -> Boolean`

Frage 20

Zu was wird folgender Ausdruck ausgewertet?

```
(myfun (myfun 2))
```

☐ `#false`

☐ 2

☐ 4

☐ Fehler

Frage 21

Die aus der Vorlesung bekannte Funktion ***`unfold`*** *hat folgende Signatur. Was macht die Funktion im dritten Argument?*

```
[X Y] (Y -> Boolean) (Y -> Y) (Y -> X) Y -> (List-of X)
```

☐ Sie erzeugt das nächste Listenelement.

☐ Sie erzeugt mit dem aktuellen Zustand ein neues Listenelement.

☐ Sie kann die Iteration terminieren.

☐ Sie gibt an, ob die Liste beendet ist.

Frage 22

Was ergibt die Auswertung des folgenden Programms?

```
(local [(define x 1)]
  (local [(define x 2)]
    3))
(+ x x)
```

☐ 2

☐ 4

☐ Fehler

☐ 3

Frage 23

Was ist syntaktischer Zucker?

☐ Konzepte einer Sprache, die für die Programmausführung unwesentlich sind.

☐ Konzepte einer Sprache, die nicht direkt einem Prozessorbefehl entsprechen.

☐ Konzepte einer Sprache, die durch andere Konzepte ausgedrückt werden können.

☐ Konzepte einer Sprache, die von einem Editor zusätzlich zum Programm angezeigt werden müssen.

Gegeben sei folgendes Programm:

```
(define (my-animate f)
  (big-bang 0
    [to-draw f]
    [on-tick ...]))
```

Frage 24

Was ist die Signatur von ***my-animate****?*

- ☐ `Number -> Number`
- ☐ `(Image -> Number) -> Image`
- ☐ `Image -> Number`
- ☐ `(Number -> Image) -> Number`

Frage 25

Wie muss ... gefüllt werden, damit ***my-animate*** *der Funktionsweise von* ***animate*** *entspricht?*

- ☐ `(lambda (x) (make-state (+ (state-tick x) 1)))`
- ☐ `(lambda (x) (x))`
- ☐ `(lambda (x) (+ x 1))`
- ☐ `(lambda (x y) (* x y))`

Frage 26

Wie kann das folgende Programm vereinfacht werden?

```
(and b1 (or (not b2) (not #t)))
```

- ☐ `(not (and b1 b2))`
- ☐ `(and b1 (not b2))`
- ☐ `(or b1 b2)`
- ☐ `(and b1 b2)`

Frage 27

Welche Funktion ist eine Funktion erster Ordnung?

- ☐ `(lambda (x) (lambda (y) y))`
- ☐ `(lambda (x) (lambda (y) (x y)))`
- ☐ `(lambda (x) ((lambda (y) y) x))`

Frage 28

Das DRY-Prinzip besagt, dass ...

- ☐ Rekursion, Schleifen und ähnliche Wiederholungen vermieden werden sollten.
- ☐ die Tastatur beim Programmieren trocken bleiben muss.
- ☐ unnötige Redundanz weitestgehend durch Abstraktion vermieden werden sollte.
- ☐ Wiederholungen im Programm ein Programm verlangsamen können.

Frage 29

Was sind magic numbers?

- ☐ Ausdrücke, die zu einer Zahl ausgewertet werden.
- ☐ Zahlen, die im Pogrammtext mehrmals auftreten.
- ☐ Zahlen wie `42`, die eine besondere Bedeutung in der Science-Fiction besitzen.
- ☐ Zahlenliterale, deren Bedeutung nicht offensichtlich ist.

Frage 30

Was ergibt die Auswertung des folgenden Programms?

```
(define X (+ 1 Y))
(define Y 6)
(* X Y)
```

- ☐ 42
- ☐ Fehler: Y wurde vor dessen Definition benutzt.
- ☐ 6
- ☐ 7

Frage 31

Was ist korrekt bei generativ rekursiven Funktionen?

- ☐ Es existiert entweder ein oder kein rekursiver Aufruf.
- ☐ Der Algorithmus terminiert nie.
- ☐ Man muss kreativer als bei struktureller Rekursion sein.
- ☐ Der Algorithmus terminiert auf eine einfache Art und Weise.

Frage 32

Was ist ein Wert der BSL?

- ☐ `(and #true 3)`
- ☐ `(define (f x) (+ 3 x))`
- ☐ `#false`
- ☐ `(+ 1 3)`

Frage 33

Laut der BSL gilt: **(war noch bisschen mehr und anders..)**

```
<e> := (<name> <e>*)
     | (cond {[<e> <e>]}+)
     | <name>
```

Was ist laut dieser Grammatik ein gültiger Ausdruck?

- ☐ `(cond [(< 5 3)] [else #false])`
- ☐ `(cond [] [(< 4 2) #true])`

Frage 34

Wenn `e1` zu `e2` reduziert werden kann, dann sagt KONG, dass ...

- ☐ `(cond e1 ...) = (cond e2 ...)`
- ☐ ...

Frage 35

Was wird in ISL bei folgender Funktion der Umgebung hinzugefügt?

```
(define f (lambda (x) (+ 2 1)))
```

- ☐ `(define f 3)`
- ☐ `(define f (lambda (x) (+ 2 1))`
- ☐ `(define f (lambda (x) 3))`
- ☐ `(define (f x) 3)`

Frage 36

Was ist die Signatur folgender Funktion?

```
(define (fun f xs) (foldr append empty (map f xs)))
```

- ☐ `[X Y] (X -> Y) (List-Of Y) -> (List-Of X)`
- ☐ `[X] (Number -> X) (List-Of Number) -> (List-Of X)`
- ☐ `[X Y] (X -> Y) (List-Of X) -> (List-Of Y)`

Der Rest ist mir/uns leider entfallen.
Grundsätzlich war die Klausur sehr ähnlich zu vorigen Klausuren.
GeTeXt von Marvin Borner.