Gedächtnisprotokoll

Praktische Informatik 1: Deklarative Programmierung
Hauptklausur

2023
Prof. Torsten Grust

Rahmenbedingungen:
90min
87 Punkte; 6 Aufgabenblöcke
30 Punkte zum Bestehen
1 doppelseitig (bedrucktes) cheat-sheet
⚠= Kennzeichen für schwere Aufgabe
Keine Signaturen oder Tests erforderlich, wenn nicht anders geschrieben

A: Signaturen (13,5P)

Im Folgenden soll auf die Signatur *any* verzichtet werden und auf parametrisierte Signaturen zurückgegriffen werden.

1. Gib die Signatur *triple* für eine 3-elementige Liste aus reellen Zahlen in monoton steigender Reihenfolge an.

```
Erfüllen Signatur:                Erfüllen Signatur nicht:
(list 0.1 3 10)                   (list 1 2 3 4)
(list 1 1 2)                      (list 12 1 4)
```

2. ⚠Gib die Signatur *list-of-bs* an, welche eine Liste mit abwechselnd erst ein Boolean und dann ein String beinhaltet.

```
Erfüllen Signatur:                 Erfüllen Signatur nicht:
(list #t "a" #f "b")               (list #t)
empty                              (list #t "d" #t)
(list #f "#f")                     (list "c" #t)
```

3. Gib die Signaturen an:

```
(list)
```

```
(list < >= >)
```

```
(lambda (xs) (map reverse xs))
```

4. Was sind parametrisierte Signaturen und welchen Vorteil haben sie?

5. Wie kann es sein, dass es homogene Listen gibt, die Elemente unterschiedlicher Datentypen beinhalten, z.B. `(list 42 "abc" #f)`

6. Gegeben sei die Signatur der Funktion f:

```
(: f ((tuple %a %b) -> %b))
```

Was berechnet f?

7. Welche Rolle spielen Signaturvariabeln in Racket?

B: Gemixte Datentypen

8. In der Informatik werden Booleans oft als 0 (= False) und 1 (= True) dargestellt. Schreibe eine Funktion *numeric*
`(: numeric (mixed boolean integer) -> integer)`,
die ein Integer oder Boolean als Input nimmt die entsprechende Zahl (also entweder den Integer oder 0 oder 1, falls ein Boolean übergeben wurde) zurückgibt.

```
(numeric #f) ⇝ 0        (numeric #t) ⇝ 1          (numeric 42) ⇝ 42
```

9. Was ist der Unterschied zwischen *enum* und *mixed*? (2P)

10. Datentypen verbrauchen Unterschiedlich viel Speicher: Folgende Liste gibt darüber Aufschluss:
    - Number: 8 Bytes
    - Boolean: 1 Bytes
    - Strings: 3 Bytes pro Buchstabe
    - Listen: 1 Byte pro Cons Zelle + Bytes des Listeninhalts
    - empty-list: 1 Byte

    Schreibe eine Funktion *size* `(: size (%a -> natural))`, die die Speichergröße einer Konstante (z.B. `(list "sef" #t 129)`) berechnet.

11. Ergänze die Lücken, damit folgende Äquivalenz gilt:

```
(filter p? (map f xs)) ≡ (map __________ (filter ______________ xs))
```

12. Was ist die Signatur von *string?* und was berechnet sie?

C: Higher Order Functions

1. Gib die Signatur von `(curry compose)` an

2. Gib die Signatur von `(curry +)` an

3. Wozu wird `((curry compose) (curry +))` ausgewertet? (Tipp: Setze zunächst die Funktionsdefinitionen ein)

4. Schreibe die Signatur und Definition der Funktion *flip*, welche eine Funktion mit zwei Argumenten konsumiert und die gleiche Funktion mit vertauschten Parametern zurückgibt.

```
(check-expect ((flip expt) 2 3) (expt 3 2)) ⇝ #t
(check-expect ((flip <) 2 3) #f) ⇝ #t
```

5. Statt `(lambda (x) (+ x 1))` kann man auch `((curry +) 1)` schreiben. Man nennt diese Schreibweise *point-free style*. Schreibe die folgenden 2 Funktionen im point-free style z.B. mithilfe curry, compose und flip um.

```
(define small?
 (lambda (x)
   (<= x 10)))

(define expt3
 (lambda (x)
   (expt x 3)))
```

6. Man kann auch Funktionen mit 3 Parametern curryen. Wie lautet die Signatur und Funktionsdefinition von *curry3* und *uncurry3*?

7. Erkläre was Endrekursion ist und was die Vorteile davon sind

8. ⚠️Schreibe eine (endrekursive) Funktion *arg-max*, die eine einstellige Funktion und eine nichtleere Liste akzeptiert und das Element der Liste zurückgibt, wofür f maximal wird.

```
(check-expect (arg-max (lambda (x) (if (even? x) 0 10))) (0 1 2 4) 1)
(check-expect (arg-max abs (list 0 -2 1)) -2)
```

D: Listenverarbeitung

1. Gegeben sei die Funktion *zip-with*:

```
(define zip-with
 (lambda (f xs1 xs2)
   (match xs1
     (empty empty)
     ((cons y ys) (match xs2
                        (empty empty)
                        ((cons y2 ys2) (cons (f y y2) (zip-with f ys ys2))))))
```

a) Wie lautet die Signatur von *zip-with*?

b) Was passiert, wenn die Listen ungleich lang sind?

2. Gegeben sei ein Ausdruck des Typs *expr*:

```
(define expr
  (signature (mixed add
                    real
                  _____)))
```

und die Record-Definition von *make-add*:

```
(define-record add
 make-add
 add?
 (num1 expr)
 (num2 expr))
```

Der Ausdruck 40 + 1 + 1 kann damit als `(make-add 40 (make-add 1 1))` geschrieben werden.

a) Definiere einen Record für Multiplikation

b) Schreibe den mathematischen Ausdruck 2 + 2 * 20 mithilfe der Records um

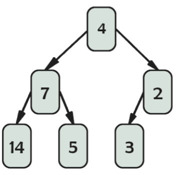
c) Schreibe eine Funktion *eval*, die einen Ausdruck des Typs *expr* auswertet `(: eval (expr -> real))`

4. Schreibe eine Funktion *all?*, die prüft, ob alle Booleans einer nichtleeren Liste #t sind.

```
(: all? ((list-of boolean) -> boolean))
```

5. Was macht die folgende Funktion *huh?*

```
(define huh?
 (lambda (xs)
   (all? (zip-with <= xs (rest xs)))))
```



E: Binärbäume

Tipp: Die Aufgaben 3 und 4 lassen sich sehr kompakt mithilfe btree-fold programmieren.

Anmerkung: In der Klausur war es ein anderer (ähnlicher) Baum.

1. Was sind die Blätter von
dem rechts abgebildeten Baum
(das ist keine Programmieraufgabe).

2. Gib die parametrisierte Signatur von *btree-fold* an.

3. Schreibe die Funktion *btree-avg*, die den Durchschnitt aller Labels eines Baums *t (btree-of real)* zurückgibt.

4. ⚠ Schreibe eine Funktion *paths*, die eine Liste mit Listen aller möglichen Wege zurückgibt (Reihenfolge egal). Für den obigen Baum würde die Funktion zurückgeben: (6P)

```
(list (list 4)
      (list 4 2)
      (list 4 2 3)
      (list 4 7)
      (list 4 7 5)
      (list 4 7 14))
```

F: Streams

1. Definiere Ausdruck *e* der in `(force (force e))` zu 42 auswertet.

2. Schreibe die Funktion inklusive Signatur *stream-zip-with*, welche einen endlosen Stream erzeugt, der über eine Funktion *f* zwei Streams miteinander verknüpft. (6P)

3. Wozu wertet folgender Term aus? (3P)

```
(stream-take 6 s)
(define s
 ((make-cell 1 (lambda () (stream-zip-with + s s)))
```

Zusammengetragen durch den Info-1 Kurs 2022/23, aufgeschrieben von Daniel Busch.