Prof. Dr. Christoph Bockisch
Fachbereich Mathematik und Informatik
AG Programmiersprachen und -werkzeuge

Philipps Universität Marburg

Übung zur Vorlesung

Deklarative Programmierung: Sommersemester 2022

Nr. 8, Abgabe bis 21.06.2022 23:55 Uhr

Aufgabe 1: (EX)TERMINATE! — 2 Punkte

(a) Geben Sie jeweils an, um welche Art von Rekursion es sich bei den folgenden Funktionen handelt (Strukturell, Generativ, Akkumulation), und begründen Sie, warum die Funktionen terminiert. (Für die lokale Funktion `sqrt/above`). **2**

```
; Nat -> Nat
; Computes the greatest natural number t with (t*t) <= x
(define (sqrt/nat x)
        (local [(define (sqrt/above x t)
                  (if (> (* t t) x)
                      (sub1 t)
                      (sqrt/above x (add1 t))))]
        (sqrt/above x 1)))
```

Hinweis: Finden Sie jeweils eine Funktion $st : T_1 \times \cdots \times T_n \to \mathbb{N}$, welche die Funktionsparameter in die natürlichen Zahlen abbildet und argumentieren Sie, dass diese mit jedem rekursiven Aufruf strikt monoton fällt.

Aufgabe 2: Nickel Cadmium — 4 Punkte

Die Fibonaccifolge ist wie folgt definiert: $fib(n) := \begin{cases} 1 & 1 \leq n \leq 2 \\ fib(n-1) + fib(n-2) & sonst \end{cases}$

(a) Implementieren Sie eine Funktion `(fib n)`, die das n-te Element der Folge nach der gegebenen Vorschrift berechnet. **1**

(b) Die Berechnung kann erheblich beschleunigt werden, indem wir zwei Akkumulatoren einführen, die jeweils $fib(n-1)$ und $fib(n-2)$ enthalten. Implementieren Sie eine Funktion `(fib/fast n)`, die das n-te Element der Folge mithilfe einer lokalen (akkumulatorgestützten) Funktion berechnet. **2**

(c) Messen Sie die Laufzeit der beiden Implementierungen für verschiedene Werte von n. Verwenden Sie dazu die Funktion `time`. Wie erklären Sie sich die erheblichen Unterschiede in der Laufzeit? **1**

Aufgabe 3: Hey, LISTEN, Mr. Freeman. Check! 2 Punkte

Implementieren Sie die folgenden Listenfunktionen:

(a) `(zip l1 l2)`, die zwei Listen zu einer Liste von Paaren zusammenführt. Die Länge der Ergebnisliste entspricht der Länge der kürzeren Listen. **1**
Beispiel: `(zip '(1 2 3) '(4 5 6))` → `'((1 4) (2 5) (3 6))`

(b) `(zip/with f l1 l2)`, die zwei Listen paarweise mithilfe der Funktion f kombiniert. Verwenden sie hierzu `foldr` und Ihre Implementierung aus Aufgabenteil a). **1**
Beispiel: `(zip/with + '(1 2 3) '(4 5 6))` → `'(5 7 9)`

(c) **Zusatzaufgabe (optional).** `(group l)`, die aufeinanderfolgende gleiche Elemente innerhalb der Liste l gruppiert. Verwenden Sie hierfür eine `fold`-Funktion. **0**
Beispiel: `(group '(1 2 2 2 3 4 5 5 6))` → `'(1 (2 2 2) 3 4 (5 5) 6)`

Aufgabe 4: Black Mesa 2 Punkte

Lösen Sie die Aufgaben unter Verwendung von Lambda-Ausdrücken. Die Benutzung lokaler Definitionen ist nicht zulässig.

Erstellen Sie die folgenden Funktionen:

(a) `; [X,Y,Z] (X Y -> Z) X -> (Y -> Z)` **1**
`(partial f x)`, die den ersten Parameter der binären Funktion f mit dem Wert x fixiert und eine neue unäre Funktion zurückliefert.

(b) `; [X] Number (X -> X) -> (X -> X)` **1**
`(repeated n f)`, die eine Funktion g zurückliefert, die f insgesamt n-mal auf ihr Argument x anwendet. D.h.: $g(x) = f^n(x) = f(f(f(...f(x)...)))$
Per Definition ist die 0-malige Anwenung einer Funktion, also $f^0(x)$ die Identitätsfunktion.

(c) **Zusatzaufgabe (optional).** `; [X,Y,Z] (X -> Y) (Y -> Z) -> (X -> Z)` **0**
`(combine f g)`, die die beiden unären Funktionen verkettet und eine neue Funktion h liefert, sodass gilt: $h(x) = g(f(x))$

Aufgabe 5: Raise! 2 Punkte

Implementieren Sie folgende Listenfunktionen mithilfe von `foldr` und Lambdas:

(a) `; [X] (X -> Boolean) (list-of X) -> (list-of X)` **1**

`(my-filter p l)`, die alle Elemente aus der Liste l ausgibt, die die Bedingung p erfüllen.

(b) `; [X] (X -> Boolean) (list-of X) -> Boolean` **1**

`(forall p l)`, die überprüft, ob alle Elemente der Liste die gegebene Eigenschaft erfüllen.

(c) **Zusatzaufgabe (optional).** `; [X] (X -> Boolean) (list-of X) -> Boolean` **0**

`(exists p l)`, die überprüft, ob mindestens ein Element der Liste die gegebene Eigenschaft erfüllt.

(d) **Zusatzaufgabe (optional).** `; [X,Y] (X -> Y) (list-of X) -> (list-of Y)` **0**

`(my-map f l)`, die jedes Element einer Liste mithilfe von f abbildet und eine Liste der abgebildeten Werte erzeugt.

Hinweis: Sie dürfen Ihre Funktionen in den folgenden Teilaufgaben verwenden.

Aufgabe 6: Schwefelhölzer

0 Punkte

Zusatzaufgabe (optional). Das Nim-2-3-Spiel[1] ist ein Spiel für 2 Personen. Zu Beginn werden Streichhölzer in 2 Reihen ausgelegt, wobei die Anzahl der Streichhölzer in den einzelnen Reihen beliebig gewählt werden kann. Die Spieler ziehen nun abwechselnd. Wer am Zug ist, entnimmt aus der Reihe, die mehr Streichhölzer enthält, 2 oder 3 Streichhölzer. Wenn beide Reihen gleich viele Streichhölzer enthalten, kann eine beliebige Reihe wählen. Wer zuerst nicht mehr ziehen kann, hat das Spiel verloren.

(a) Schreiben Sie eine Funktion: **0**

```
; Number Number -> Bool
; Checks if one can win the game if the two rows contain
; n and m matches.

(check-expect (win? 2 2) false)
(check-expect (win? 3 2) false)
(check-expect (win? 2 4) true)
(check-expect (win? 2 5) true)
(check-expect (win? 9 6) true)

(define (win? n m) ...
```

[1] Der Name stammt aus dem Alt-Englischen *to nim* - stehlen. Es gibt diverse Varianten des Nim-Spiels, siehe `https://de.wikipedia.org/wiki/Nim-Spiel`

`(win? n m)` berechnet, ob der Spieler, **der gerade am Zug ist**, das Spiel gewinnen kann, wenn in den 2 Reihen n und m Streichhölzer liegen.

(b) Zeigen Sie, dass Ihre `(win? n m)`-Implementierung terminiert. **0**

(c) Schreiben Sie eine Funktion: **0**

```
; Number Number -> String
; Suggests a move for the pick-a-match game

(check-expect (suggest 3 2) "Du kannst das Spiel mit keinem Zug mehr
↪  gewinnen")
(check-expect (suggest 2 2) "Du kannst das Spiel mit keinem Zug mehr
↪  gewinnen")
(check-expect (suggest 6 2) "Ziehe 3 aus der ersten Reihe")
(check-expect (suggest 4 2) "Ziehe 2 aus der ersten Reihe")
(check-expect (suggest 2 4) "Ziehe 2 aus der zweiten Reihe")
(check-expect (suggest 2 5) "Ziehe 3 aus der zweiten Reihe")

(define (suggest m n) ...
```

`(suggest m n)` soll dabei dem Spieler den nächsten Zug empfehlen, sodass dieser gewinnt. Greifen Sie dabei auf ihre `(win? n m)`-Funktion zurück. Ist es nicht möglich für den Spieler zu gewinnen, soll ein Text mit dieser Information zurückgegeben werden.