Christian-Albrechts-Universität zu Kiel
Institut für Informatik
Lehrstuhl für Programmiersprachen und Übersetzerkonstruktion
Prof. Dr. M. Hanus, F. Reck

Institut für Informatik

12. Übung zur Vorlesung „Informatik I“

Abgabe am Freitag, 30.1.2009, 14 Uhr

In dieser Serie geht es um Datenströme. Die in der Vorlesung besprochenen Operationen auf Datenströmen werden Ihnen über die Übungswebsite als Teachpack `stroeme.scm` zur Verfügung gestellt und stehen in iLearn zur Verfügung. Teachpacks können Sie in DrScheme über den Menüpunkt *Language → Add Teachpack* einbinden.
Das Teachpack enthält die Funktionen `liste->strom` und `strom->liste`, die das Erzeugen bzw. die Ausgabe von Strömen vereinfachen.

Präsenzaufgabe 1

5 Philosophen sitzen um einen runden Tisch - jeder mit einer Schüssel Reis vor sich. Zwischen je zwei Reisschüsseln liegt ein Essstäbchen. Wenn ein Philosoph mit dem Denken fertig ist und hungrig wird, greift er sich zunächst das Stäbchen (`st-links`) links von seiner Reisschüssel, danach das rechte (`st-rechts`) und beginnt zu essen. Wenn er fertig ist, legt er die Stäbchen in der selben Reihenfolge wieder ab, in der er sie genommen hat. Wird eines der Essstäbchen bereits von einem Tischnachbar verwendet, muss der hungrige Philosoph warten, bis dieser es wieder zurückgelegt hat. Die dinierenden Philosophen kann man in Scheme mit Semaphoren (`sem-links` und `sem-rechts`) durch folgende Funktion modellieren:

```
(define (essen phil st-links st-rechts sem-links sem-rechts)
  (begin
    (P sem-links)
    (P sem-rechts)
    (reis-essen phil st-links st-rechts)
    (V sem-links)
    (V sem-rechts)))
```

(a) Können zwei Philosophen gleichzeitig das selbe Stäbchen benutzen?

(b) Kann es passieren, dass einige oder sogar alle Philosophen verhungern?

(c) Ist es bei diesen Beobachtungen von Bedeutung, dass es gerade 5 Philosophen sind?

(d) Wie könnte man sicherstellen, dass immer ein hungriger Philosoph essen kann?

Präsenzaufgabe 2

- Definieren Sie die beiden folgenden Funktionen zur Berechnung von Strömen:

  (a) `(replicate x n)` soll zu *(x x x ...)* mit Länge `n` und
  (b) `(scanl f z` *(x1 x2 x3 ...)*`)` zu *(z (f z x1) (f (f z x1) x2) ...)*

  ausgewertet werden.

- Definieren Sie eine Funktion `(inits strom)`, die den Strom aller Anfangsstücke von `strom` berechnet. Zum Beispiel soll der Ausdruck `(inits` *(0 1 2)*`)` zu *(() (0) (0 1) (0 1 2))* ausgewertet werden.
- Definieren Sie den Strom der ersten `n` Quadratzahlen ohne die Multiplikation zu verwenden. Versuchen Sie, Funktionen aus den anderen Aufgabenteilen wiederzuverwenden.

Aufgabe 3 4 Punkte

Implementieren Sie eine Operation `(kombiniere-mit op s1 s2)`, die zwei Ströme elementweise zu einem neuen zusammensetzt. Zum Beispiel soll der Aufruf

`(kombiniere-mit +` *(1 2 3 4) (4 3 2 1)*`)`

zum Strom *(5 5 5 5)* ausgewertet werden. Für Ihre Implementierung können Sie davon ausgehen, dass die beiden Ströme gleich lang sind.

Aufgabe 4 6 Punkte

Implementieren Sie eine Operation `(akk-n op e s)`. Diese Operation ähnelt der Operation `akk` aus der Vorlesung, nur operiert sie auf einem Strom von Strömen, statt nur auf einem Strom. `akk-n` verwendet nacheinander die Akkumulationsprozedur zur Kombination aller ersten Elemente der Eingabeströme, aller zweiten Elemente usw. Hat einer der Eingabeströme keine weiteren Elemente, so werden die restlichen Elemente in den anderen Datenströmen ignoriert. Das Ergebnis ist ein Datenstrom der Ergebnisse dieser Anwendungen. Beispiel: Für den Strom

`s =` *((1 2 3 4) (5 6 7) (8 9 10 11 12) (13 14 15 16))*

soll `(akk-n + 0 s)` zu dem Strom *(27 31 35)* ausgewertet werden.

Aufgabe 5 10 Punkte

Zeichnen Sie die Umgebungen, Bindungen und Prozeduren, die bei der Auswertung von

```
(define (konstr-flag)
  (local
     ((define flag false)
      (define (set b) (set! flag b))
      (define (get) flag)
      (define (zuteilen m)
        (cond
          ((eq? m 'set) set)
          ((eq? m 'get) get))))
    zuteilen))

(define a (konstr-flag))
(define b (konstr-flag))
((a 'set) true)
((a 'get))
((b 'get))
```

entstehen.