Grundlagen der Informatik 1
WS 2008/09

Prof. Mühlhäuser, Dr. Rößling, Melanie Hartmann, Daniel Schreiber
http://proffs.tk.informatik.tu-darmstadt.de/gdi1

Übung 7 - Lösungsvorschlag Version: 1.0 01.12.2008

---

1 Mini Quiz

Streams

- ☒ sind nur möglich bei normaler Auswertungsreihenfolge.
- ☐ sind nur möglich bei applikativer Auswertungsreihenfolge.
- ☐ erzeugen einen Datenstrom fester Länge.
- ☐ besitzen eine feste Abbruchbedingung.

Interpreter

- ☒ Interpreter können die Semantik der interpretierten Sprache selbst definieren.
- ☐ Interpreter können die Semantik der Basissprache selbst definieren.
- ☒ Bei einem Meta-Interpreter gleichen sich die interpretierte und die Basis-Sprache.
- ☒ Es existieren Programmiersprachen, die ausschließlich durch ihren Interpreter definiert sind.

2 Fragen

1. Wann kann die Verwendung von Zustandsvariablen sinnvoll sein? Gibt es Fälle, in denen eine Zustandsvariable unbedingt benötigt wird?

   Lösungsvorschlag:
   Die Verwendung von Zustandsvariablen kann sinnvoll sein, wenn mehrere Funktionen auf die gleichen Objekte zugreifen müssen. Mit Hilfe einer Zustandsvariable müssen diese Objekte nicht mehr überall als Parameter übergeben werden. Jedes Programm lässt sich so umformen, dass keine Zustandsvariablen mehr benötigt werden. T10.24

2. Welche Probleme bringen Zustandsvariablen mit sich?

   Lösungsvorschlag:
   Sie öffnen einen zusätzlichen Kommunikationskanal, der Informationsfluss ist schwerer zu kontrollieren. Es erschwert vorherzusagen, wann welche Variable geändert wird, da dies auch nicht im Vertrag erfasst wird.

3. Was ändert sich am Programmiermodell durch die Einführung von Zustandsvariablen?

Lösungsvorschlag:
Skript T10.14

- *Wichtige Invarianten (referentielle Transparenz, Konfluenz) gelten nicht mehr*
- *Die Reihenfolge der Auswertung wird zum entscheidenden Faktor*
- *Der Begriff der Identität wird eingeführt*

3 Zustandsvariablen und Objekte

Zustandsvariablen haben viele oft unerwünschte Nebeneffekte (s. Fragen). Objektorientierte Programmiersprachen versuchen teilweise den "Schaden", der durch Zustandsvariablen angerichtet wird, zu begrenzen, indem sie Kommunikationsbarrieren aufbauen: die Kommunikation über Zustandsvariablen kann immer nur innerhalb eines Objekts stattfinden. Objekte kann man auch in Scheme implementieren. Betrachten Sie den folgenden Code:

```
1  ;; contract:
2
3  ;; purpose:
4
5  (define (make-new-obj)
6    (local (
7      (define state 0))
8      (lambda (method params)
9        (cond
10         [(symbol=? 'set method)
11          ((lambda (p1) (set! state p1)) (first params))]
12         [(symbol=? 'get method) state]
13         [else (error "message not understood")])))))
```

1. Was liefert der Aufruf `(make-new-obj)`? Geben Sie Vertrag und Beschreibung an.

Lösungsvorschlag:
Eine Funktion mit Gedächtnis.

Vertrag: make-new-obj: ->(symbol list-of-X ->Y)

Beschreibung: gibt ein Objekt in Form einer Funktion mit Gedächtnis zurück

2. Was tut `(define o (make-new-obj))`?

Lösungsvorschlag:
Die Definition bindet eine Funktion mit Gedächtnis an den Namen o.

3. Welche Operationen unterstützen mit `make-new-obj` erzeugte Objekte und wie werden Sie aufgerufen?

Lösungsvorschlag:
Die Operationen 'set und 'get werden unterstützt. Aufruf erfolgt mit `((make-new-obj) 'set (list 5))` *bzw.* `((make-new-obj) 'get empty)`

4. Können Operationen, die mit einem Objekt $o1$ durchgeführt werden, einen Einfluss haben auf die Operationen, die mit $o2$ durchgeführt werden?

   Lösungsvorschlag:
   Nein, das ist nicht möglich.

5. Ergänzen Sie `make-new-obj` um eine Methode `add`, die den übergebenen Parameter (eine Zahl) zu dem im Objekt gespeicherten Zustand addiert.

6. Ergänzen Sie `make-new-obj` um eine Methode `add2`, die zwei Parameter (zwei Zahhlen) übernimmt und beide zu dem im Objekt gespeicherten Zustand addiert.

   Lösungsvorschlag:

```
;; contract: make-new-obj: -> (symbol list-of-X -> Y)
;; purpose: returns an object in form of a function with memory
(define (make-new-obj)
  (local (
    (define state 0))
    (lambda (method params)
      (cond
        [(symbol=? 'set method)
         ((lambda (p1) (set! state p1)) (first params))]
        [(symbol=? 'add method)
         ((lambda (p1) (set! state (+ state p1))) (first params))]
        [(symbol=? 'add2 method)
         ((lambda (p1 p2) (set! state (+ state p1 p2))) (first params
           ) (first (rest params)))]
        [(symbol=? 'get method) state]
        [else (error "message not understood")]))))
```

7. Der Interpreter legt die Semantik der interpretierten Sprache fest. Ändern Sie die Semantik so, dass die Parameter für die Methode *add2* in umgekehrter Reihenfolge übergeben werden müssen!

   Lösungsvorschlag:

```
;; contract: make-new-obj: -> (symbol list-of-X -> Y)
;; purpose: returns an object in form of a function with memory
(define (make-new-obj)
  (local (
    (define state 0))
    (lambda (method params)
      (cond
        [(symbol=? 'set method)
         ((lambda (p1) (set! state p1)) (first params))]
        [(symbol=? 'add method)
         ((lambda (p1) (set! state (+ state p1))) (first params))]
        [(symbol=? 'add2 method)
         ((lambda (p1 p2) (set! state (+ state p1 p2))) (first (rest
           params)) (first params))]
        [(symbol=? 'get method) state]
        [else (error "message not understood")]))))
```

8. In der Vorlesung wurde der Datentyp `map` vorgestellt (s. T9.18ff). Den wollen wir nun ähnlich als Objekt kapseln.

   Definieren Sie eine lokale Zustandsvariable `map-data`. Schreiben Sie die Operationen `extend`, `remove` und `lookup` von `map` so um, dass kein Parameter vom Typ `map` mehr erforderlich sind, sondern stattdessen die lokale Zustandsvariable `map-data` verwendet wird.

   Zur Erinnerung: die beiden Funktionen von `map` wurden wie folgt definiert:

```
;; contract: map-extend: symbol X (mapof X) -> (mapof X)
;; purpose: extend the given map base-env with the new key-value pair
   (var, val) and return the new map
(define (map-extend var val base-env)
  (cons (make-binding var val) base-env))

;; contract: map-remove: symbol (mapof X) -> (mapof X)
;; purpose: remove the element with the key name from the list a-map
(define (map-remove name a-map)
  (filter (lambda (b) (not (symbol=? name (binding-name b))))
          a-map))

;; contract: map-lookup: symbol (mapof X) -> X or empty'
;; purpose: lookup the elements that are stored for the key name in
  the map a-map
(define (map-lookup name a-map)
  (if (empty? a-map)
      empty
      (if (symbol=? name (binding-name (first a-map)))
          (binding-value (first a-map))
          (map-lookup name (rest a-map)))))
```

   Speichern Sie, statt eine neue `map` zurückzuliefern, den neuen Wert mit `set!` in `map-data`. Bei der Implementierung von `lookup` müssen Sie eine lokale Hilfsfunktion verwenden. Orientieren Sie sich am Code zu Teilaufgabe 1.

Lösungsvorschlag:

```
;; contract: make-new-map:-> (symbol list-of-X -> Y)
;; purpose: returns a map object

(define (make-new-map)
  (local (
     (define-struct binding (name value))
     (define map-data empty))

     (lambda (method params)
      (cond
        [(symbol=? 'extend method)
         ((lambda (name value) (set! map-data (cons (make-binding
           name value) map-data))) (first params) (first (rest params
           ))))]
        [(symbol=? 'remove method)
         ((lambda (name) (set! map-data (filter
                                  (lambda (b) (not (symbol=? name (
                                    binding-name b))))
                                  map-data))) (first params))]
        [(symbol=? 'lookup method)
```

```
18          ((lambda (name)
19            (local (
20                    (define (h name m)
21                       (if (empty? m)
22                           empty
23                           (if (symbol=? name (binding-name (first m))
                                 )
24                               (binding-value (first m))
25                               (h name (rest m))))))
26          (h name map-data))) (first params))])))
```

4 Streams

Streams eignen sich zur Darstellung unendlich langer Zahlenfolgen. Wegen ihrer unendlichen Größe können diese nicht als Listen dargestellt werden. In Übung 5 haben Sie das Newton Verfahren zur Approximation von Nullstellen einer Funktion kennengelernt.
Zur Erinnerung: Das Newton Verfahren arbeitet rekursiv: Ausgehend von einem Startwert x_n wird eine bessere Schätzung x_{n+1} wie folgt berechnet:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

1. Definieren Sie eine Funktion `next-newton` für die Funktion $f(x) = x^2$ (Hinweis: $f'(x) = 2x$), die einen Startwert entgegennimmt und davon ausgehend die nächst bessere Schätzung zurückliefern.

   Lösungsvorschlag:

```
1  ;; contract: num -> num
2  ;; purpose: computes the next approximation of a zero of
3  ;;          x*x following Newton's method.
4  ;; example: (next-newton 3) -> 1.5
5  (define next-newton (lambda (last-newton)
6          (- last-newton (/ (* last-newton last-newton)
7              (* 2 last-newton)))))
```

2. Die Funktion `next-newton` findet nur Nullstellen von $f(x) = x^2$. Die Implementierung kann aber leicht so geändert werden, dass Nullstellen für andere Funktionen gefunden werden. Diese Änderungen wollen wir nicht für jede Funktion f von Hand durchführen, sondern wir wollen eine höherwertige Funktion `make-next-newton` implementieren. `make-next-newton` erhält eine Funktion f und ihre Ableitung f' (entspricht der Funktion `df`) als Parameter und liefert die entsprechende Variante von `next-newton` für diese Funktion.

   Der Aufruf

```
1          (make-next-newton (lambda (x) (* x x)) (lambda(x)(* 2 x)))
```

   erzeugt zum Beispiel eine Funktion die äquivalent zu `next-newton` von oben ist.

   Lösungsvorschlag:

```
;;contract: (num -> num) (num -> num) -> (num -> num)
;;purpose: generates a function for computing the next approximation
;;          of a zero of f following Newton's method.
;;example: (make-next-newton (lambda (x) (* x x)) (lambda(x)(* 2 x)))
(define make-next-newton (lambda (f df)
     (lambda (last-newton)
          (- last-newton (/ (f last-newton) (df last-newton))))))
```

3. Implementieren Sie die Funktion `newton-approx`, die eine Funktion f, ihre Ableitung f' und einen Startwert x_0 übergeben bekommt und einen Stream von Approximationen der Nullstelle liefert.

   Lösungsvorschlag:

```
;; contract: newton-approx: (num -> num) (num-> num) num ->
;;                                                  stream-of-numbers
;; purpose: returns a stream of approximations of a zero of f
(define newton-approx (lambda (f df start-value)
      (repeat (make-next-newton f df) start-value)))
```

4. Implementieren Sie die Funktion `closeValue`, die einen Toleranzwert epsilon eps, eine Funktion f und einen Stream von Zahlen l konsumiert. Die Funktion soll denjenigen Zahlenwert x des Stroms zurückliefern, für den der Betrag des Funktionswerts $f(x)$ zum ersten Mal unterhalb der Toleranzschwelle eps liegt, d.h. x nahe genug an der gesuchten Nullstelle liegt.
   Lösungsvorschlag:

```
;;contract: closeValue: num (num -> num) stream-of-num -> num
;;purpose: returns the first value in the given stream that with
;;          |(f x)|<eps

(define closeValue (lambda (eps f l)
    (if (< (abs (f (my-first l))) eps)
      (my-first l)
      (closeValue eps f (my-rest l)))))
```

5. Warum funktioniert das Programm nicht mit DrScheme? Warum funktioniert es im Interpreter mit normaler Auswertungsreihenfolge aus der Vorlage?

   Lösungsvorschlag:
   DrScheme verwendet die applikative Ausertungsreihenfolge. Aus diesem Grund versucht DrScheme jeden Wert des Streams zu berechnen. Da der Stream aber unendlich lang ist, terminiert das Programm nie. Der Interpreter aus der Vorlesung hingegen verwendet die normale Auswertungsreihenfolge, die Elemente des Streams werden erst dann berechnet, wenn sie wirklich gebraucht werden.

6. Starten Sie das Programm im Interpreter, der vom GdI1-Portal runtergeladen werden kann, und finden Sie eine Nullstelle der Funktion $f(x) = cos(x)$ (Hinweis: $f'(x) = -sin(x)$). Dazu müssen Sie den Interpreter um die Funktion cos und sin als primitive Funktionen erweitern.

   Lösungsvorschlag:
   Der Interpreter kann folgendermaßen um `sin` *und* `cos` *erweitert werden:*

```
(define primitive-procedures
  (local
    ;;added sin and cos as names for ...
    ((define names
            '(first rest cons list empty? empty * / - + < = abs sin
              cos))
     ;;... builtin sin and cos function
    (define procs
        (list first rest cons list empty? empty * / - + < = abs sin
          cos)))
    (map-create names (map make-primitive-procedure procs))))
```

Hausübung

Die Vorlagen für die Bearbeitung werden im GdI1-Portal bereitgestellt. Kommentieren Sie Ihren selbst erstellten Code. Die Hausübung muss bis zum Abgabedatum im GdI1-Portal abgegeben werden. Der Fachbereich Informatik misst der Einhaltung der Grundregeln der wissenschaftlichen Ethik großen Wert bei. Zu diesen gehört auch die strikte Verfolgung von Plagiarismus. Mit der Abgabe Ihrer Hausübung bestätigen Sie, dass Sie bzw. Ihre Gruppe alleiniger Autor des gesamten Materials sind. Falls Ihnen die Verwendung von Fremdmaterial gestattet war, so müssen Sie dessen Quellen deutlich zitieren.
textbfAbgabedatum: Dieses Mal müssen Sie keine Hausaufgaben abgeben.

5 WebTasks

In diesem Übungsblatt haben Sie die Möglichkeit, Ihre Java-Fähigkeiten schon einmal mit dem WebTasks-System zu üben. Diese Übung wird dringend empfohlen. Es findet keine Bepunktung durch den Tutor statt, da die Korrektur vom WebTasks System automatisch vorgenommen wird.

5.1 Anmeldung am System und erste Schritte

1. Rufen Sie das WebTasks-System im Browser unter folgender URL auf:
   `http://webtasks.informatik.tu-darmstadt.de/webtasks/`
2. Geben Sie Ihren RBG Benutzernamen (den gleichen, den Sie im Portal verwenden) ein.
3. Geben Sie Ihr RBG Passwort (das gleiche wie im Portal) ein.
4. **Setzen Sie den Haken bei "RBG Account"!**
5. Klicken Sie auf Login.

Sie sind jetzt angemeldet und können Aufgaben bearbeiten. Um eine Aufgabe zu bearbeiten, gehen Sie wie folgt vor:

1. Suchen Sie sich eine Aufgabe aus der Liste aus und klicken Sie auf Details.
2. Klicken Sie auf den Reiter "Lösung einreichen".
3. Die Aufgabenstellung und Randbedingungen (etwa dass keine Rekursion verwendet werden darf) werden angezeigt.

4. Sie finden hier auch Eingabefelder vor, in die Sie ihren Java Code eingeben können.
5. Um die Lösung abzugeben, klicken Sie auf "Abschicken".
6. Ihre Lösung wird dann automatisch getestet.

Bei diesem Test können zwei Dinge passieren: Ihre Lösung besteht alle Testfälle und wird als korrekt angesehen. In dem Fall können Sie nun auch die Lösung anderer Teilnehmer zu dieser Aufgabe ansehen und kommentieren. Die zweite Möglichkeit ist, dass die Lösung nicht korrekt ist. Ist dies der Fall, zeigt WebTasks eine Fehlermeldung an, ähnlich zu denen, die Sie von `check-expect` Testfällen aus DrScheme kennen. **Sie können dann Ihre Lösung so lange verbessern und erneut einreichen, bis alle Testfälle durchlaufen!**

5.2 Empfohlene Aufgaben

Sie können gerne beliebige Aufgaben aus dem WebTask System bearbeiten. Wir empfehlen die folgenden Aufgaben zu bearbeiten, zumindest eine aus jedem Bereich. Zu den von uns empfohlenen Aufgaben können Ihnen auch Ihre Tutoren helfen. Der Schwierigkeitsgrad der Aufgaben ist deutlich geringer als etwa bei den Hausübungen zu Scheme.
Die Aufgabenseiten erreichen Sie direkt unter dem Link
`http://webtasks.informatik.tu-darmstadt.de/webtasks/uploadsolution.jsp?taskid=XXX`,
wobei XXX durch die unten bei der Aufgabe angegebene Nummer zu ersetzen ist.

1. Berechnungen durchführen
   - Arrays & Schleifen - Check Sum: 112
2. Berechnungen auf Arrays durchführen
   - Arrays & Schleifen - Average: 82
   - Arrays & Schleifen - Element Near To Average: 87
   - Arrays & Schleifen - Count Elements By Number: 84
   - Arrays & Schleifen - Count Elements By Even Odd: 83
3. Arrays bearbeiten oder neu erstellen
   - Arrays & Schleifen - Append: 81
   - Arrays & Schleifen - Filter Even Or Odd: 88
   - Arrays & Schleifen - Find Doubles: 144
4. Arbeiten mit Matrizen
   - Arrays & Schleifen - Matrix Plus Matrix: 157
   - Arrays & Schleifen - Matrix Transponieren: 158
5. Schreiben einer Klasse (mit einer statischen Methode)
   - Rekursion - Minimum: 22

5.3 Wie geht es weiter?

Sie können die WebTasks Datenbank auch während des Semesters und auch später im Studium zum Üben benutzen. Sie können auch Freunde auf WebTasks hinweisen - selbst wenn diese (noch) nicht an der TUD studieren. Das Angebot steht allen registrierten Nutzern offen.