Grundlagen der Informatik 1
WS 2008/09

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

Übung 5 vv1.0 17.11.2008

---

1 Mini Quiz

Kreuzen Sie die wahren Aussagen an!

1. ☐ lambda-Ausdrücke sollten verwendet werden, wenn eine Prozedur nicht rekursiv ist und nur einmal als Argument einer anderen Prozedur gebraucht wird.
2. ☐ Strukturell rekursive Prozeduren terminieren naturgemäß.
3. ☐ Prozeduren mit Gedächtnis können nur mit `lamda` Ausdrücken erzeugt werden.
4. ☐ Generative Rekursion ist effizienter als strukturelle Rekursion.
5. ☐ Jede strukturell rekursive Funktion ist auch generativ rekursiv.

2 Fragen

1. Was ist der Unterschied zwischen generativer und struktureller Rekursion? Nennen Sie für jede Art von Rekursion ein Anwendungsbeispiel.
2. Welche Vorgehensweise verfolgt ein Backtracking-Algorithmus?

3 Generative vs. strukturelle Rekursion

Die in der Vorlesung vorgestellte Vorlage für rekursive Algorithmen sieht wie folgt aus:

```
1 (define (recursive-fun problem)
2   (cond
3     [(trivially-solvable? problem)
4      (determine-solution problem)]
5     [else
6      (combine-solutions
7       problem
8       (recursive-fun (generate-problem problem)))]))
```

1. Für welche Art von Rekursion ist diese Vorlage gedacht?

2. Die Funktion `recursive-fun` soll folgenden Vertrag haben: `recursive-fun : (listof X) -> number`. Sie soll die Länge der übergebenen Liste berechnen. Ändern Sie dazu nicht die Definition von `recursive-fun`, sondern definieren Sie diese vier Funktionen auf geeignete Weise:
   - `trivially-solvable?`
   - `determine-solution`
   - `combine-solutions`
   - `generate-problem`
3. Welche Art von Rekursion haben Sie letztendlich benutzt?

4 Das Newton Verfahren (K)

Mit dem Newton Verfahren lässt sich näherungsweise eine Nullstelle einer Funktion f berechnen. Die Funktion f muss einigen Bedingungen genügen, die hier nicht weiter erörtert werden sollen. Das Newton Verfahren arbeitet rekursiv: Ausgehend von einer Schätzung x_n wird eine bessere Schätzung x_{n+1} wie folgt berechnet:

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

Das Verfahren startet mit einer beliebigen initialen Schätzung x_0. Das Verfahren wird abgebrochen, sobald die Änderung von x_n zu x_{n+1} kleiner einer Schranke δ ist. Die letzte Schätzung x_{n+1} wird dann als Näherungswert für die Nullstelle zurückgegeben. Schreiben Sie die Prozedur `newton-method`, die

- eine Funktion f (`f`)
- deren Ableitung f' (`dfx`)
- einen Startwert x_0 (`x`)
- und eine Schranke δ (`delta`)

erhält und näherungsweise eine Nullstelle der Funktion f mit Hilfe des Newton Verfahrens bestimmt. Geben Sie zunächst Vertrag, Beschreibung und ein Beispiel für die Prozedur an. Implementieren Sie dann die Prozedur.

5 Zahldarstellung (K)

Schreiben Sie eine Funktion `convert: num num -> lon`, die eine Zahl x zur Basis b darstellt. Unter der Darstellung einer Zahl x zur Basis b verstehen wir eine Liste von Zahlen $a_{n-1}...a_0$ mit

$$x = \sum_{i=0}^{n-1} a_i \cdot b^i, a_i \in \mathbb{Z}, 0 <= a_i < b$$

$n \in \mathbb{N}$ ist dabei die kleinste Potenz von b, die größer oder gleich x ist, d.h. für die $x \leq b^n$ gilt, n ist also die Anzahl der Stellen von x in der gesuchten Darstellung.

Beispiel: Soll $x = 10$ im Binärsystem $b = 2$ dargestellt werden gilt: n ist gleich 4, da $2^4 > 10$ und $2^3 < 10$. $10 = 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0$. Die Liste, die zurückgegeben werden soll ist (a_3, a_2, a_1, a_0), also `'(1 0 1 0)`.

1. Schreiben Sie eine Funktion `lengthRepresentation: num num -> num`, die x und b konsumiert und die Anzahl der Stellen der resultierenden Darstellung (n) zurückgibt. Sie können die Funktion `expt: num num -> num` verwenden. `(expt x y)` liefert x^y.
   Beispiel: `(lengthRepresentation 10 2)` ergibt 4.

2. Schreiben Sie nun die Funktion `convert`. Sie können folgende in Scheme eingebaute Funktionen verwenden:
   - `expt`, s. oben
   - `floor: num -> num`, rundet eine Zahl ab: `(floor 3.7)` ist 3.
   - `remainder: num num -> num`, gibt den Rest der Division der beiden übergebenen Zahlen zurück: `(remainder 10 3)` ist 1.
   - verwenden Sie nicht `append` sondern nur `cons`!

Hausübung

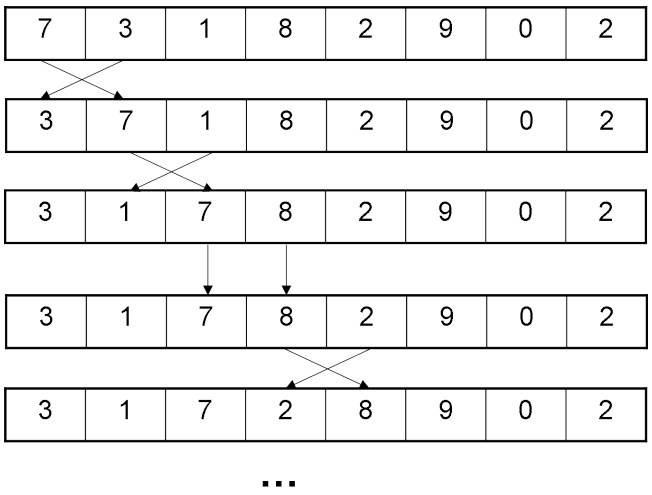
Die Vorlagen für die Bearbeitung werden im GdI1-Portal bereitgestellt. Kommentieren Sie Ihren selbst erstellten Code. Die Hausübung muss bis zum Abgabedatum im GdI1-Portal abgegeben werden. Der Fachbereich Informatik misst der Einhaltung der Grundregeln der wissenschaftlichen Ethik großen Wert bei. Zu 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.

Abgabedatum: Fr, 28.11.08, 16:00 Uhr.

Denken sie daran ihren Code mindestens mit Verträgen und Beschreibungen zu kommentieren, sowie für jede Prozedur 2 Testfälle anzugeben. Wählen sie für Hilfsfunktionen und Parameter sinnvolle Namen. Benutzen sie als Sprachlevel "Zwischenstufe mit Lambda".

6 Bubblesort (6 P)

Ein Sortierverfahren für Listen von Zahlen ist BubbleSort. Beim BubbleSort Verfahren wird eine Liste in mehreren Durchläufen sortiert. Bei einem BubbleSort Durchlauf werden alle in der Liste benachbarten Elemente verglichen. Stehen zwei benachbarte Elemente nicht in der richtigen Reihenfolge, werden Sie vertauscht. Das Bild zeigt ein Beispiel für einen BubbleSort Durchlauf.



Hinweis: Sei $(a_0\, a_1\, a_2 \ldots a_n)$ die zu sortierende Liste. Im ersten Schritt werden a_0 und a_1 betrachtet. Das kleinere der beiden Elemente a_0 und a_1 nennen wir s das andere l. Die Liste sieht nach dem ersten Schritt so aus: $(s\, l\, a_2 \ldots a_n)$. Im nächsten Schritt muss dann l mit a_2 verglichen werden, usw.

1. Überlegen Sie sich ein geeignetes Abbruchkriterium. Wann ist ein BubbleSort Durchlauf für eine Liste trivial und wie sieht das Ergebnis in diesen trivialen Fällen aus? Beanworten Sie diese Frage **in Worten** mit einem Satz!

2. Ergänzen Sie die Funktion `bubblesort-run` aus der Vorlage. Ergänzen Sie auch den Vertrag! `bubblesort-run` konsumiert eine Liste von Zahlen, führt einen BubbleSort-Durchlauf auf dieser Liste durch und gibt das Ergebnis des Durchlauf zurück. Verwenden Sie als Rekursionsanker den von ihnen in der ersten Teilaufgabe gefundenen trivialen Fall. **Beispiel:** `(bubblesort-run '(1 3 2 4 2))` ergibt `'(1 2 3 2 4)`.

Nach jeden Durchlauf BubbleSort steht das größte Element am Ende der Liste, der vordere Teil der Liste ist noch unsortiert. Der BubbleSort Algorithmus führt nun so lange BubbleSort Durchläufe auf dem unsortierten Teil der Liste durch, bis die ganze Liste sortiert ist.

3. Ergänzen Sie die Funktion `bubblesort` aus der Vorlage. Ergänzen Sie auch den Vertrag! `bubblesort` konsumiert eine Liste von Zahlen und gibt sie nach dem BubbleSort Algorithmus sortiert zurück. Sie dürfen die Funktionen `last` und `head` aus der Vorlage verwenden. `last` konsumiert eine Liste und gibt deren letztes Element zurück. `head` konsumiert eine Liste, schneidet das letzte Element ab und gibt die so verkürzte Liste zurück. Beispiel: `(head '(a b c))` ergibt `'(a b)`.

7 Pythagoras-Baum (7 P)

7.1 Turtle Grafik

In dieser Übung soll in Scheme mit den `turtle` Zeichenbefehlen gezeichnet werden. Dazu muss das Teachpack `turtle.ss` installiert werden. Hierzu rufen Sie in der Menüleiste die Option "Sprache → Teachpack hinzufügen..." auf. Sollte das Teachpack nicht in der Liste stehen, finden Sie es im DrScheme Installationsverzeichnis unter collects/teachpack/turtle.ss.
Der Befehl `(turtles)` erzeugt eine Turtle-Instanz. Sie können dem Turtle Befehle erteilen. Alle Turtel Befehle werden in DrScheme zu `(void)` ausgewertet haben aber einen Nebeneffekt: Sie zeichnen

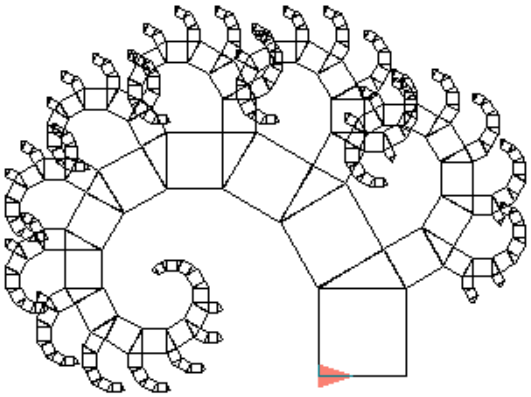
auf dem Zeichenfenster. Um dem Turtle mehrere Befehle zu erteilen, reicht es diese in eine Liste zu schreiben, `(list (draw 10) (move 5))` wird z.B. zu `(list (void) (void))` ausgewertet, als Nebeneffekt wird aber eine Linie gezeichnet und der Turtle bewegt.
Die folgende Liste zeigt alle Befehle, die notwendig sind, um die Turtle zu bewegen und zu zeichnen:

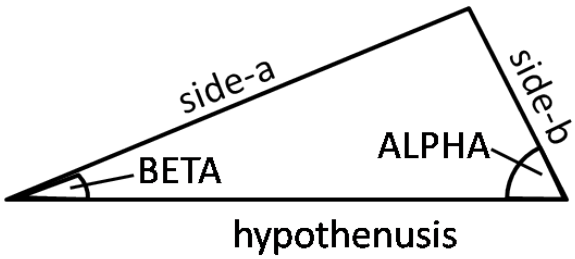
Turtle-Befehl	Semantik
`(draw n)`	Linie der Länge n in aktuelle Richtung zeichnen
`(move n)`	n Schritte in aktuelle Richtung gehen
`(turn a)`	Um a Grad nach links drehen
`(turn -a)`	Um a Grad nach rechts drehen

7.2 Pythagoras-Baum Zeichnen

Nicht nur Funktionen können rekursiv definiert werden, sondern auch Punktemengen. Ein Beispiel dafür ist der rekursiv definierte Pythagoras-Baum. Einen Pythagoras-Baum zeichnet man folgendermaßen:

- Zeichne ein Quadrat mit Seitenlänge a
- Zeichne auf der Oberseite des Quadrats ein rechtwinkliges Dreieck mit Basiswinkeln α und β. Die Länge der Hypothenuse des Dreiecks ist a
- Dies ist der Pythagoras-Baum der Stufe n. Wiederhole diese beiden Schritte auf den beiden Katheten des Dreiecks, a ist dann jeweils die Länge der Katheten. Dies ist der Pythagoras-Baum der Stufe $n + 1$.





1. Schreiben Sie die Funktion `pythagoras-step`, die die Seitenläng a konsumiert und die ersten beiden Schritte beim Zeichnen eines Pythagoras-Baum durchführt. Zum Zeichnen des Dreiecks können Sie die Funktion `paintTriangle` aus der Vorlage verwenden. Diese konsumiert eine Zahl a und zeichnet ein rechtwinkliges Dreieck mit der Hypothenusenlänge a. `paintTriangle` verwendet die ebenfalls in der Vorlage definierten Winkel `ALPHA` und `BETA`. Alle Funktionen in ihrer Abgabe müssen mit Vertrag, Beispiel, Beschreibung und mindestens zwei Testfällen versehen sein!

2. Überlegen Sie sich ein geeignetes Abbruchkriterium, das bestimmt bis zu welcher Stufe ein Pythagorasbaum gezeichnet wird. Beanworten Sie diese Frage **in Worten** mit einem Satz!

3. Schreiben Sie die Funktion `pythagoras-tree` die einen Pythagoras-Baum zeichnet. Verwenden Sie das von Ihnen erdachte Abbruchkriterium. Zur Berechnung der neuen Seitenlänge a können Sie die Funktionen `get-a` und `get-b` verwenden, die die Hypothenusenlänge eines rechtwinkligen Dreiecks konsumieren und jeweils die Länge einer Kathete zurückgeben.