Grundlagen der Informatik 1

Wintersemester 2009/2010

Prof. Dr. Max Mühlhäuser, Dr. Rößling
http://proffs.tk.informatik.tu-darmstadt.de/teaching

Übung 5 Version: 1.0 16. 11. 2009

---

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 `lambda` 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 soll folgenden Vertrag haben: recursive-fun: (listof X) -> number. Sie soll die Länge der übergebenen Liste berechnen. Ändern Sie *nicht* die Definition von `recursive-fun`, sondern definieren Sie die folgenden 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, indem ausgehend von einer Schätzung x_n wie folgt eine bessere Schätzung x_{n+1} berechnet wird:

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

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 eine 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: number number -> (listof number)`, 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: Für die Darstellung von $x = 10$ im Binärsystem mit $b = 2$ gilt $n = 4$, da $2^4 > 10$ und $2^3 < 10$. Für $10 = 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0$ soll die zurückzugebende Liste die Form (a_3, a_2, a_1, a_0) haben, entspricht also `'(1 0 1 0)`.

1. Schreiben Sie eine Funktion `length-representation: number number -> number`, die x und b konsumiert und die Anzahl der Stellen der resultierenden Darstellung (n) zurückgibt. Sie können die Funktion `expt: number number -> number` verwenden. `(expt x y)` liefert x^y.

   Beispiel: `(length-representation 10 2)` ergibt 4.

2. Schreiben Sie nun die Funktion `convert`. Sie können folgende in Scheme eingebaute Funktionen verwenden:
   - `expt`, s. oben
   - `floor: number -> number` rundet eine Zahl ab: `(floor 3.7)` ist `3`.
   - `remainder: number number -> number` 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. Falls Sie die Hausübung in einer Lerngruppe bearbeitet haben, geben Sie dies bitte deutlich bei der Abgabe an. Alle anderen Mitglieder der Lerngruppe müssen als Abgabe einen Verweis auf die gemeinsame Bearbeitung einreichen, damit die Abgabe im Portal auch für sie bewertet werden kann.

Abgabedatum: Freitag, 27. 11. 2009, 16:00 Uhr

Denken Sie bitte daran, Ihren Code hinreichend gemäß den Vorgaben zu kommentieren (Scheme: Vertrag, Beschreibung und Beispiel sowie zwei Testfälle pro Funktion; Java: JavaDoc). Zerlegen Sie Ihren Code sinnvoll und versuchen Sie, wo es möglich ist, bestehende Funktionen wiederzuverwenden. Wählen Sie sinnvolle Namen für Hilfsfunktionen und Parameter.
Benutzen sie als Sprachlevel "Advanced Student".

6 Selection Sort (3 Punkte)

Selection Sort ist eines der naheliegendesten Sortierverfahren. Hierbei wird in jedem Schritt aus der betrachteten (Rest-)Liste das kleinste Element (*min-value*) herausgesucht und an das Ende der bisherigen Ergebnisliste gesetzt, gefolgt vom sortierten Rest der Liste - natürlich ohne *min-value*.
Spielen Sie zunächst das Verfahren anhand der obigen Beschreibung mit einer kurzen Liste von Zahlen auf Papier durch.
In dieser Aufgabe implementieren Sie eine *universal einsetzbare Form* von Selection Sort.

1. Implementieren Sie eine Funktion remove−**first** zur Entfernung eines Elements aus einer Liste. Die Funktion konsumiert den zu entfernenden Wert–vom Typ *X*–, einen Vergleichsoperator eq−x?: X X −> boolean sowie eine Liste von Elementen des Typs *X*. Das Ergebnis ist eine Liste vom Typ *X* ohne das *erste* Auftreten des zu entfernenden Werts (1 Punkt)!

   Hinweis: Im Template finden Sie zwei konkrete Tests, die Ihre Prozedur passieren sollte. Zur Erleichterung sind Vertrag, Beschreibung und Beispiel sowie Tests bereits angegeben.

2. Implementieren Sie die Funktion *selection-sort*, die eine Liste von Elementen von Typ *X* und *zwei* Funktionen der Form X X −> boolean bzw. X X −> X konsumiert.

   Die erste Funktion testet zwei Elemente vom Typ X auf Gleichheit und liefert entsprechend **true** oder **false**. Die zweite Funktion bestimmt das *kleinere* der beiden übergebenen Elemente und produziert daher ein Element von Typ X. Details finden Sie auch in den Vorlagen, inklusive zwei Beispieltests (2 Punkte).

   Hinweis: Es macht Sinn, den kleinsten Wert der Liste lokal zu definieren und dann "passend" einzusetzen. Mit *foldl* lässt sich die Suche nach dem kleinsten Wert sehr elegant lösen.

   Hinweis: Wir werden Ihr Sortierverfahren nicht nur mit Zahlen, sondern auch mit Strukturen und verschiedenen Ordnungsrelationen darauf testen. Achten Sie daher darauf, sauber mit den Vergleichsoperatoren und nicht mit den eingebauten Operationen wie $=$, $<$, $<=$ zu arbeiten!

7 Zeichnen von Booleschen Bäumen (6 Punkte)

Die Struktur der booleschen Bäume haben Sie bereits in Übungsblatt 3 kennengelernt. Diese Struktur soll nun mittels eines in DrScheme enthaltenen Teachpacks visualisiert werden.
Installieren Sie hierzu das Teachpack `draw.ss` über den Menüpunkt "Sprache → Teachpack hinzufügen ...". Eine Übersicht für das `draw.ss` Teachpack finden Sie unter `http://download.plt-scheme.org/doc/372/html/teachpack/draw.html`.
Das `draw.ss` Teachpack stellt unter anderem Prozeduren zum Zeichnen verschiedener grafischer Primitive bereit. Für diese Übung relevant sind:

- start : number number −> void erzeugt und initialisiert ein Fenster mit der übergebenen Größe,
- draw−solid−line: posn posn −> void zeichnet eine Linie,
- draw−solid−string: posn string −> void zeichnet einen String an die angegebene Position.

Die im Teachpack genutzte Struktur posn entspricht unserer Struktur *point* und besteht aus den beiden Feldern *x* und *y*.
Hinweis: Für die Bearbeitung dieser Aufgabe benötigen Sie Ihre Lösung (oder unsere Vorgaben dazu) zur Hausübung 3 "Boolesche Bäume", da diese als Eingabe für diese Aufgabe dienen. Stellen Sie daher sicher, dass Sie diesen Code bereitliegen haben!
Gegeben sei ein boolescher Baum, bekannt aus der Hausaufgabe von Übungsblatt 3. Zur Erinnerung: die Blätter eines booleschen Baums sind immer Instanzen von *bool-direct*, innere Knoten sind entweder Instanzen von *bool-unary* (nur ein Nachfolger) oder von *bool-binary* (zwei Nachfolger). Ein gegebener geschachtelter Listenausdruck wird durch die Methode input−>tree in einen booleschen Baum transformiert.
Hinweis: Vor der ersten Zeichenoperation müssen Sie einmal start aufrufen, um die Zeichenumgebung zu initialisieren.
Die Ausgabe soll ähnlich wie in Abbildung 1 aussehen. Allerdings braucht Ihr Text *nicht* zentriert an der entsprechenden Position zu stehen!

Abbildung 1: Darstellung eines booleschen Baums

1. Implementieren Sie eine Scheme-Prozedur bool−direct−>string: bool−direct −> string. Diese konsumiert eine Struktur vom Typ *bool-direct* (siehe Übungsblatt 3) und produziert einen String. Achten Sie darauf, alle möglichen Werte in den jeweils passenden String zu verwandeln (1 Punkt)!

   Hinweis: Verwenden Sie **symbol**−>string: **symbol** −> string, um aus einem Symbol eine Zeichenkette zu erzeugen.

2. Schreiben Sie eine Scheme-Prozedur tree−depth: bool−tree −> number. Diese konsumiert einen booleschen Baum und produziert die Tiefe des Baums, die definiert ist als die Länge der längsten Knotenfolge von der Wurzel zu einem Blatt (1 Punkt).

   Hinweis: Die Tiefe des Baumes in der Abbildung ist 3.

3. Implementieren Sie nun eine Scheme-Prozedur

   render−tree: bool−tree number number number −> void

   Diese konsumiert einen booleschen Baum, eine x- und y-Koordinate sowie eine initiale Breite konsumiert und zeichnet den booleschen Baum. Dabei soll die Wurzel des Baumes mit dessen Wert an den angegebenen Koordinaten gezeichnet werden. Der linke Teilbaum soll um die gegebene Breite nach links und unten, der rechte Teilbaum entsprechend um die gegebene Breite nach rechts und unten versetzt werden. Für tiefer liegende Äste soll die Breite jeweils halbiert werden (siehe Abbildung 1) (2 Punkte).

   Hinweis: Verwenden Sie **begin**, um mehrere Zeichenbefehle in einer Sequenz zusammenzufassen. So führt der folgende *begin*-Ausdruck der Reihe nach die Ausdrücke *expression1, expression2,* ... aus und gibt als Ergebnis den Wert des letzten Ausdrucks zurück:

```
(begin
  expression1
  expression2
  ...)
```

   Hinweis: Auf Tests können Sie verzichten; stattdessen sollten Sie Ihre Funktion mit einigen booleschen Bäumen aufrufen und das Ergebnis optisch überprüfen.

4. Für ein konsistenteres Erscheinungsbild der verschiedenen booleschen Bäume ist die initiale Breite so berechnen, dass die *untersten* Äste mit einer vorgegebenen Breite gezeichnet werden. Schreiben Sie dazu eine Funktion determine− initial −width: bool−tree number −> number, welche für einen gegebenen booleschen Baum und eine Breite der Äste berechnet, mit welcher initialen Breite render−tree aufgerufen werden muss (1 Punkt).

   Hinweis: Aus der Vorlesung kennen Sie bereits fast−expt. Diese Prozedur kann dabei helfen, aus einer minimalen Breite der untersten Äste die initiale Breite zu berechnen.

```
;; fast-expt: number number -> number
;; Raise b to its nth power
(define (fast-expt b n)
  (cond
    [(= n 0) 1]
    [(even? n) (sqr (fast-expt b (/ n 2)))]
    [else (* b (fast-expt b (- n 1)))]))
```

5. Verstecken Sie die Prozedur render−tree als lokale Prozedur render−tree−inner in einer Prozedur render−tree−minimal−width: bool−tree number number number −> void. Diese konsumiert als letztes Argument die Breite der untersten Äste und verhält sich ansonsten wie die ursprüngliche Prozedur (1 Punkt).