Grundlagen der Informatik 1

Sommersemester 2011

Dr. Guido Rößling
https://moodle.informatik.tu-darmstadt.de/

Übung 5 Version: 1.0 16. 05. 2011

---

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 recursive-fun soll den Vertrag recursive-fun: (listof X)-> **number** haben und 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 dazu 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 als eine 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)—als **lambda**-Ausdruck—
- deren Ableitung f' (dfx)—ebenfalls als **lambda**-Ausdruck—
- 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 ganze Zahl, für die gilt $b^n \geq x$. n ist also die Anzahl der Stellen von x in der gesuchten Darstellung.
Beispiel: Soll $x = 10$ im Binärsystem $b = 2$ dargestellt werden, so 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 hat die Form (a_3, a_2, a_1, a_0), 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 von Racket bereitgestellte 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 Lernportal Informatik bereitgestellt. Kommentieren Sie Ihren selbst erstellten Code. Die Hausübung muss bis zum Abgabedatum im Lernportal Informatik 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 Lernportal Informatik auch für sie bewertet werden kann. Beachten Sie dazu die Hinweise bei der Aufgabenabgabe im Lernportal Informatik!

Abgabedatum: Freitag, 27. 05. 2011, 16:00 Uhr

Denken Sie bitte daran, Ihren Code hinreichend gemäß den Vorgaben zu kommentieren (Racket: Vertrag, Beschreibung und Beispiel sowie zwei Testfälle pro Funktion; Vertrag, Beschreibung und Beispiel für jede **local** definierte Funktion; Vertrag (ohne Namen) und kurze Beschreibung für jeden **lambda**-Ausdruck; 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 für diese Übung als Sprachlevel "Advanced Student". Zuweisungen sind aber **explizit nicht gestattet** *und führen zu Punktabzug!*

6 Berechnen der Fußball-Bundesligatabelle (10 Punkte)

In dieser Aufgabe widmen wir uns einem weiteren praktischen— und wenn nicht ganz alltäglichem, so doch fast *wöchentlichem*—Problem: dem Berechnen der Fußball-Bundesligatabelle.

6.1 Definition der Datenstrukturen

Wir betrachten zunächst die Datentypendefinitionen. Ein *Fußballspiel* (soccer−match) enthält die Namen der beiden Mannschaften und das Ergebnis. Dabei steht der Wert gh für die Anzahl Tore der Heimmannschaft und der Wert gg für die Anzahl Tore der Gastmannschaft.

```
1  ;; soccer−match − represents a single sports match (e.g., for soccer)
2  ;; home: String − name of the home team
3  ;; guest: String − name of the guest team
4  ;; gh: number [>=0] − goals for the home team
5  ;; gg: number [>=0] − goals for the guest team
6  (define−struct soccer−match (home guest gh gg))
7
8  ;; score−entry − represents a single sports score entry
9  ;; team: String − name of the team
10 ;; gf: number [>=0] − goals scored by the team
11 ;; ga: number [>=0] − goals scored against the team
12 ;; diff: number − goal difference (gf − ga)
13 ;; points: number − points received so far
14 ;; (for soccer, can be only 0, 1 or 3 for a single match)
```

```
15 (define-struct score-entry (team gf ga diff  points))
16
17 ;; A score is either empty or a (list  f a p), where f, a, p are numbers
18 ;; for the goals for (f) and against (a) the team, and the points received (p).
```

Die Datenstruktur score-entry repräsentiert einen Eintrag, wie er in der Bundesligatabelle zu finden ist. Jeder Eintrag umfasst den Namen der Mannschaft, die erzielten Tore, die erhaltenen Tore, die Tordifferenz (erzielte minus erhaltene Tore) und die insgesamt erreichten Punkte. Für jeden Sieg gibt es dabei drei, für ein Unentschieden einen und für eine Niederlage null Punkte. Bitte beachten Sie, dass diese Datenstruktur sowohl zur Repräsentierung *eines* Spiels (aus Sicht einer einzelnen Mannschaft) als auch für das Gesamtergebnis einer Mannschaft nach n Spielen genutzt werden kann.
Die informelle Definition von score ist entweder *empty* oder eine Liste mit den Angaben der Tore der erzielten bzw. erhaltenen Toren sowie der Punktzahl. Mit dieser Definition können Sie die ständige Neudefinition von Instanzen von score-entry vermeiden.
Wie üblich definiert die Vorlage einige Spielergebnisse, die Sie für Ihre Tests und Beispiele nutzen können.
Zu beachten: Wir stellen Ihnen *umfangreiche* Tests im Template zur Verfügung. Sie erhalten daher keinen Punktabzug, wenn Sie keine zusätzlichen Tests verfassen—es ist aber natürlich sinnvoll, dies hier und da zu tun!

6.2 Von Ihnen zu implementierende Prozeduren (10 Punkte)

Bitte implementieren Sie auf Basis der Vorlage die folgenden Prozeduren. Dabei sollten Sie weitere Hilfsprozeduren auf Basis des Wunschlistenansatzes identifizieren und implementieren. Bitte deklarieren Sie die von Ihnen definierten Hilfsprozeduren **local**, *nachdem* Sie sie hinreichend getestet haben. Hilfsprozeduren, die von mehr als einer Prozedur genutzt werden, lassen Sie bitte global definiert, um den Code nicht kopieren zu müssen.
Die folgenden Prozeduren *müssen* für unsere Tests—auch zu Ihren Gunsten, um Fehler leichter eingrenzen zu können—*global sichtbar*, also nicht ineinander geschachtelt in einem **local**, implementiert werden:

- create-goals-point-entry: **String** soccer-match -> score. Diese Prozedur konsumiert den Namen einer Mannschaft und ein einziges Spielergebnis. Sie liefert einen *score*, d.h. eine Liste der Länge 3, mit der korrekten Angabe der von der Mannschaft im Spiel erzielten Tore, gefolgt von der Anzahl erhaltener Tore, und zuletzt der dafür vergebenen Punkte.

  Hinweis: Falls die Mannschaft im Spiel nicht vertreten war, ist (**list** 0 0 0) als Ergebnis zu liefern, *auf keinen Fall ein Fehler*.

  Zu beachten: Die einzelnen Mannschaften treten im Laufe der Spieltage mal als Heim- und mal als Gastmannschaft auf. Achten Sie darauf, dass Sie einer Mannschaft—etwa „Eintracht Frankfurt"—immer die korrekte Anzahl erzielter bzw. erhaltener Tore und entsprechend Punkte zuteilen! Eine Unterscheidung zwischen „Mannschaft X ist Heimmannschaft" und „... ist Gastmannschaft" wäre sicherlich sinnvoll.

  Für diese Methode erhalten Sie bis zu zwei Punkte.

- determine-points-for-team: **String** (listof soccer-match)-> score-entry. Diese Prozedur konsumiert den Namen *einer* Mannschaft und *alle* Begegnungen. Sie berechnet den Tabelleneintrag für die Mannschaft nach Auswertung *aller* Partien.

  Für diese Methode erhalten Sie bis zu drei Punkte.

- score-is-better-than?: score-entry score-entry -> **boolean**. Diese Prozedur vergleicht zwei Instanzen von score-entry und liefert genau dann **true**, wenn der erste Eintrag „besser" als der zweite Eintrag ist. Eine genaue Definition, wann ein Eintrag „besser" als ein anderer ist, finden Sie in Abschnitt 6.3.

Für diese Methode erhalten Sie bis zu zwei Punkte.

- sort−soccer−table: (listof **String**)(listof soccer−match)−> (listof score−entry). Diese Prozedur konsumiert die Namen *aller* Mannschaften (als Liste von Strings) sowie die Ergebnisse *aller* Spiele. Sie liefert eine *sortierte* Liste von Instanzen von score−entry gemäß den Regeln in Abschnitt 6.3.

  Verwenden Sie für die Sortierung eine angepasste Fassung von insertion−sort (aus Folie T5.57-59), indem Sie die dort in der Prozedur insert „fest codierte" Nutzung von <= durch einen Aufruf von score−is−better−than? ersetzen. Ersetzen Sie zusätzlich das in den Folien verwendete fold durch **foldr** oder **foldl** und beachten Sie die geänderte Reihenfolge der Parameter.

  Für die korrekte Implementierung von sort−soccer−table erhalten Sie bis zu drei Punkte, von denen einer auf die Anpassung von insertion−sort entfällt.

Bei dieser Aufgabe werden Ihnen bewusst einige Freiheiten gelassen, damit Sie selbst erproben können, wie eine geeignete Modellierung aussehen kann. Es gibt viele unterschiedlich(e) gute Ansätze!

6.3 Wichtige Erläuterungen zum Vergleich von Mannschaften

Bitte lesen Sie die folgenden Hinweise *aufmerksam*, selbst wenn Sie ein Experte für Fußball sind (oder sich dafür halten). Im Folgenden wird **verbindlich** beschrieben, nach welchen Kriterien Ihre Methode score−is−better−than? die „bessere" von zwei Mannschaften bestimmen muss.
Zu beachten: In der Abschlusstabelle steht die *beste* Mannschaft immer oben, bei uns also am Anfang der Liste.
Gegeben seien zwei Einträge der Struktur score−entry namens entryA und entryB. Beide Einträge liefern, wie oben beschrieben, Informationen zur Mannschaft, die erzielten und erhaltenen Tore, die Tordifferenz und die Punkte der jeweiligen Mannschaft.
Für Vergleich der beiden Mannschaften ist *in exakt der folgenden Reihenfolge zu verfahren*:

1. Die Mannschaft mit *mehr Punkten* steht in der Ergebnisliste vor der Mannschaft mit weniger Punkten. Eine Mannschaft mit 5 Punkten ist also immer „besser" als eine Mannschaft mit weniger als 5 Punkten.

2. Bei *gleicher* Punktzahl steht die Mannschaft mit der *besseren Tordifferenz*—die negativ sein kann—vor der Mannschaft mit der schlechteren Tordifferenz. Eine Mannschaft mit Tordifferenz -3 ist bei gleicher Punktzahl also „besser" als eine Mannschaft mit Tordifferenz -4.

3. Bei gleicher Punktzahl *und* gleicher Tordifferenz steht die Mannschaft mit den mehr erzielten Toren vor der Mannschaft mit weniger erzielten Toren. So käme etwa eine Mannschaft mit jeweils 3 erzielten und erhaltenen Toren (und damit mit Tordifferenz 0) vor einer Mannschaft mit jeweils zwei erzielten und erhaltenen Toren (ebenfalls Tordifferenz 0), da mehr (3) Tore erzielt wurden als von der anderen Mannschaft (2).

4. Sind Punktzahl, Tordifferenz *und* Anzahl erzielter Tore gleich—was in der Realität insbesondere an den ersten Spieltagen durchaus passiert!—, werden die Mannschaften *alphabetisch* aufsteigend angeordnet. Verwenden Sie dazu **string**−ci<=?: **String String** −> **boolean**.