Ludwig-Maximilians-Universität München
Department "Institut für Informatik"
Prof. Dr. Christian Böhm
Elke Achtert, Arthur Zimek

München, 15.12.2006

Informatik 1
WS 2006/07

Übungsblatt 9: Typisierung, Polymorphie, Currying, Komplexität

Besprechung: 08.01.–12.01.2007

Abgabe aller mit **Hausaufgabe** markierten Aufgaben bis Freitag, 22.12.2006, 18:00 Uhr

Aufgabe 9-1 *Typisierbarkeit* **(Hausaufgabe)**

Geben Sie in einer Datei `9-1.txt` für jede der folgenden Funktionen entweder den Typ mit einer Begründung an, wie er zustandekommt, oder eine Begründung, warum die Funktion keinen Typ hat.

(a) `fun a(x) = if x>=0 then x else ~1.0`

(b) `fun b(x,y) = if x then y+1 else x`

(c) `fun c(x) = c(x=x)`

(d) `fun d(x) = if true then d(x) else d(x)=d(x)`

Aufgabe 9-2 *Typisierung, Polymorphie, Currying* **(Hausaufgabe)**

Definieren Sie in einer Datei `9-2.sml` eine Funktion mit Namen `f1` und Typ
`'a * 'b * ('a * 'b -> 'c) -> 'a * 'b * 'c`
sowie eine Funktion mit Namen `f2`, die die curried Form von `f1` ist.

Aufgabe 9-3 *Currying, Polymorphie* **(Hausaufgabe)**

In einem Ausblick im 1. Kapitel haben wir die Funktionen `max` und `groesstesElement` bereits polymorph definiert:

```
val max = fn (x, y, ord) =>
    if ord(x, y) then x
    else y;

val rec groesstesElement =
    fn (L, ord) =>
    if length(L) = 1 then hd(L)
    else max(hd(L),
             groesstesElement(tl(L), ord),
             ord);
```

Inzwischen haben Sie alles nötige gelernt, um diese Definition vollständig zu verstehen.

(a) Geben Sie in einer Datei `9-3.sml` `max` und `groesstesElement` in curried Form an, so dass gilt:
`val max = fn : ('a * 'a -> bool) -> 'a -> 'a -> 'a`
und
`val groesstesElement = fn : ('a * 'a -> bool) -> 'a list -> 'a`.

(b) Geben Sie in der Datei `9-3.sml` nun zusätzlich möglichst kompakt eine Funktion `groesstesElementInt` vom Typ `int list -> int` an, die eine Liste von Integern auf das größte Element in dieser Liste abbildet.

(c) Die Datei `9-3c.sml` enthält Farben und Werte für Spielkarten wie folgt:

```
val Farbe = ["Herz", "Karo", "Kreuz", "Pik"];
val Wert  = ["7", "8", "9", "10", "Bube", "Dame", "Koenig", "Ass"];
```

Definieren Sie in der Datei `9-3.sml` einen Typ `Karte` für ein Tupel `(Farbe,Wert)` von zwei Strings. `Karte` ist also eine Abkürzung für den Typ `string * string`.

Hinweis: Eine `Karte` kann nun immer noch aus beliebigen Strings gebildet werden. Wir werden bald eine Möglichkeit kennenlernen, den Wertebereich eines neuen Typs besser einzugrenzen.

Definieren Sie dann eine Liste `Spiel` vom Typ `Karte list`, die alle möglichen Karten eines Spieles enthält, also die Menge `Farbe`×`Wert` für die Mengen `Farbe` und `Wert` aus Datei `9-3c.sml`.

Definieren Sie eine Ordnungs-Relation für die Menge `Spiel`, durch eine Prädikat-Funktion
`ordKarten : Karte * Karte -> bool`.
`ordKarten(karte1,karte2)` soll `true` ergeben, wenn der Wert von `karte1` größer ist als der Wert von `karte2`, oder, falls beide Werte gleich sind, wenn die Farbe von `karte1` größer oder gleich der Farbe von `karte2` ist. Ein Wert bzw. eine Farbe ist umso größer, je weiter rechts sie in der Liste `Wert` bzw. `Farbe` steht.

Hinweis: Wenn Sie beim Erstellen der Definitionen das Gefühl bekommen, das sei eine stupide Arbeit, ist sehr wahrscheinlich eine elegantere Lösung möglich.

Aufgabe 9-4 *Listen und Polymorphie, Komplexität* **(Hausaufgabe)**

Das Sortieren von Listen ist ein wichtiges Problem der Informatik, zu dem sehr unterschiedliche Lösungen entwickelt wurden.

(a) Der Algorithmus *Insertion Sort* betrachtet für eine gegebene Liste alle Elemente nacheinander. Jedes Element x wird in eine neue, zweite Liste vor dem ersten Element y eingefügt, für das gilt: $x < y$. Wenn es kein solches Element y gibt, wird x am Ende der zweiten Liste eingefügt. Wenn das nächste Element (der Nachfolger von x in der Eingabeliste) betrachtet wird, enthält die zweite Liste bereits alle Elemente der Eingabeliste bis zu x (einschließlich). Wenn x das erste Element ist, ist die zweite Liste also leer. Die zweite Liste ist das Ergebnis des Algorithmus. Implementieren Sie dieses Verfahren in Datei `9-4a.sml` in einer SML-Funktion
`insertionSort : int list -> int list`

Hinweis: Es wird relativ einfach, wenn Sie eine Hilfsfunktion definieren, die eine einzelne Zahl an der richtigen Stelle in eine bereits sortierte Liste einfügt.

(b) Welches asymptotische Laufzeitverhalten hat dieses Verfahren? Wie lange braucht es mindestens (bester Fall) und längstens (schlechtester Fall)? Wie sehen diese Fälle aus? Was kann man als durchschnittlichen Zeitbedarf betrachten?

Geben Sie Ihre Überlegungen in einer Datei `9-4b.txt` ab.

Hinweis: Bestimmen Sie die Laufzeitkomplexität aufgrund der Anzahl der benötigten Vergleiche von einzelnen Elementen.

(c) Verallgemeinern Sie in einer Datei `9-4c.sml` die Funktion (in curried Form) als ein polymorphes Verfahren für eine als Parameter gegebene Ordnung. Die Funktion hat dann die Signatur:

```
insertionSortPolymorph : ('a * 'a -> bool) -> 'a list -> 'a list
```

(d) Der Algorithmus *Merge Sort* verfolgt eine andere Strategie:

- Um eine Liste zu sortieren, wird sie in zwei Hälften geteilt, die unabhängig voneinander sortiert werden. (Diese Strategie nennt man *Divide and Conquer*[1] — nach dem lateinischen Strategem *divide et impera*. Sie beruht darauf, dass ein kleineres Problem häufig wesentlich einfacher zu lösen ist als ein gleichartiges größeres.)
- Anschließend werden die beiden sortierten Listen verschmolzen (*merge*), so dass die Sortierung erhalten bleibt. Man bildet also eine neue Liste, in der jeweils das kleinere der beiden führenden Elemente der beiden sortierten Teillisten ausgewählt wird. An dieses wird die Liste angehängt, die aus den beiden Teillisten gebildet wird, nachdem dieses Element entfernt wurde.
- Eine leere Liste oder eine einelementige Liste ist bereits sortiert. (Hier wird also tatsächlich das kleinere Problem sehr einfach.)

Implementieren Sie dieses Verfahren in Datei `9-4d.sml` in einer SML-Funktion
`mergeSort : int list -> int list`

Hinweis: Hilfreiche Funktionen aus der SML Standard-Bibliothek hierfür sind:
`List.length`, `List.take` und `List.drop`.

(e) Welches asymptotische Laufzeitverhalten hat dieses Verfahren? Geben Sie Ihre Überlegungen in einer Datei `9-4e.txt` ab.

Hinweis: Bestimmen Sie die Laufzeitkomplexität aufgrund der Anzahl der benötigten Vergleiche von einzelnen Elementen.

(f) Verallgemeinern Sie in einer Datei `9-4f.sml` die Funktion (in curried Form) als ein polymorphes Verfahren für eine als Parameter gegebene Ordnung. Die Funktion hat dann die Signatur:
`mergeSortPolymorph : ('a * 'a -> bool) -> 'a list -> 'a list`

[1] nicht zu verwechseln mit *Command and Conquer*