Ludwig-Maximilians-Universität München
Institut für Informatik
Prof. Dr. M. Wirsing, M. Hammer

SoSe 2009
Übungsblatt 4

Programmierung und Modellierung

Aufgabe 4-1 Applikative Auswertung, Sonderausdrücke

a) Die folgende Funktion prüft – etwas umständlich – ob eine ganze Zahl tatsächlich eine ganze Zahl ist:

```
fun ganz(x)   =   x=0  orelse  ganz(abs(x)-1);
```

Diese Funktion bildet jeden `int`-Wert auf den Wahrheitswert `true` ab. Dass man sogar dazu Rekursion verwenden kann, hat Hiasl Hirndübl ausgetüftelt.

Beschreiben Sie in einer Datei `4-1a.txt` mit dem Substitutionsmodell, wie der Ausdruck `ganz(~1)` in SML ausgewertet wird. Geben Sie genügend viele Zwischenschritte an, damit unmissverständlich klar wird, welche Teilausdrücke in welcher Reihenfolge ausgewertet werden und welchen Wert sie jeweils haben.

b) Bartl Bastscho bastelt sich seine eigene Disjunktion, weil er sich den komischen Namen `orelse` so schlecht merken kann:

```
fun or(false, false)   =   false
  | or(false, true )   =   true
  | or(true,  false)   =   true
  | or(true,  true)    =   true;
```

Er testet die Funktion erfolgreich mit Beispielen wie `or( 5<3, 5>=3 )` und ist gerade dabei, sie als Infixoperator zu definieren, als Vroni Frogstmi vorbeikommt und sagt: „Also die Funktion vom Hiasl würde damit überhaupt nicht terminieren. Du hast ja genau die Wahrheitstafel implementiert. In der Vorlesung haben wir aber gehört, dass `orelse`-Ausdrücke in SML nicht mit der Wahrheitstafel ausgewertet werden, sondern als Sonderausdrücke."

Bartl erinnert sich nur undeutlich, aber er ändert seine Definition. Zum Testen baut er sein `or` in die Funktion von Hiasl ein. Er ist verblüfft, als er merkt, dass sie auch so nicht terminiert:

```
fun or(A1,A2)   =   A1  orelse  A2;
fun ganz'(x)    =   or( x=0,  ganz'(abs(x)-1) );
```

Beschreiben Sie in einer Datei `4-1b.txt` mit dem Substitutionsmodell, wie der Ausdruck `ganz'(~1)` in SML ausgewertet wird.

Aufgabe 4-2 Programmieren in SML

Schreiben Sie eine Funktion `strToList(s)`, die einen String `s` in eine Liste seiner Zeichen umwandelt. Verwenden Sie dazu nur die Funktionen, die auf Seite 11 des Foliensatzes 4 angegeben wurden. Geben Sie Ihre Implementierung in einer Datei `4-2.sml` ab.

Hinweis: Oftmals benötigen Sie mehr Parameter, um eine Rekursion sinnvoll durchführen zu können. Hier empfielt es sich, eine Hilfsfunktion zu definieren, in der die eigentliche Arbeit erledigt wird. Bei dieser Aufgabe benutzt ein möglicher Lösungsweg so eine Hilfsfunktion, die zusätzlich zu dem String noch die aktuelle Position übergeben bekommt. Es ist jedoch auch möglich, ohne Hilfsfunktion auszukommen.

Aufgabe 4-3 Listen

In dieser Aufgabe sollen einige Funktionen auf Listen definiert werden. Geben Sie diese Funktionen in einer Datei `4-3.sml` ab.

a) Schreiben Sie eine SML-Funktion `zaehle(l, e)`, welche in einer Liste `l` die Anzahl der Vorkommen vom Element `e` zählt. So soll z.B. der Aufruf `zaehle([1,2,3,4],5)` den Wert `0` ergeben, und der Aufruf `zaehle(["a","b","a","c"], "a")` den Wert 2.[1]

b) Schreiben Sie eine SML-Funktion `entferne(l,e)`, die alle Elemente gleich `e` aus der Liste `l` entfernt. So soll beispielsweise `entferne([1,2,3,1,4], 1) = [2,3,4]` gelten.

c) Schreiben Sie eine SML-Funktion `duplikate(l)`, die die Anzahl der Elemente ausgibt, die in der Liste `l` mehrfach vorkommen. So soll z.B. der Aufruf von `duplikate([1,2,3,2,1])` den Wert `2` und der Aufruf von `duplikate([1,2,1,3,1])` den Wert `1` zurückgeben (die `1` wird nur einfach gezählt). Sie dürfen (und sollten) dabei auf bereits von Ihnen definierte Funktionen zurückgreifen.

Abgabe: Montag, den 25.5.2009, 12 Uhr, per UniWorx.

[1] Neuere SML-Implementierungen werden Ihre Implementierung mit der Warnung „`calling polyEqual`“ kommentieren. Dies ist kein wirkliches Problem: der Algorithmus ist nur etwas ineffizient, siehe http://www.cs.williams.edu/~freund/cs334-083/examples/ml/ML-hints.html.