Technische Universität München
Fakultät für Informatik

A. Berlea
berlea@in.tum.de
Raum: MI 02.07.053

Programmiersprachen

Wintersemester 2006/2007

3. Übungsblatt *16. November 2006*

Aufgabe 1:

Definiere folgende Funktionen!

a) `map : ('a -> 'b) -> 'a list -> 'b list`
`map f l` wendet `f` auf jedes Element aus `l` an und liefert die Liste der Ergebnisse zurück.

b) `find : ('a -> bool) -> 'a list -> 'a option`
`find f l` wendet `f` auf Elemente `x` der Liste `l` von links nach rechts solange an, bis `f x` den Wert `true` zurückliefert; `SOME x` wird zurückgeliefert, wenn ein solches `x` existiert, sonst `NONE` (Der Typ `'a option` ist wie folgt definiert: `datatype 'a option = NONE | SOME of 'a`).

c) `filter : ('a -> bool) -> 'a list -> 'a list`
`filter f l` wendet `f` auf jedes Element `x` aus `l` an und liefert die Liste aller `x` zurück, für welche `f x` den Wert `true` hat.

d) `partition : ('a -> bool) -> 'a list -> 'a list * 'a list`
`partition f l` wendet `f` auf jedes Element aus `l` und liefert ein Paar `(pos,neg)` zurück, wobei `pos` die Liste jener `x` ist, für welche `f x` den Wert `true` zurückliefert, und `neg` die Liste jener `x`, für welche `f x` den Wert `false` zurückliefert.

e) `foldr : ('a * 'b -> 'b) -> 'b -> 'a list -> 'b`
`foldr f b [x1, x2, ..., xn]` liefert `f(x1, f(x2, ..., f(xn, b)...))` oder `b`, wenn die Liste leer ist, zurück.

f) `foldl : ('a * 'b -> 'b) -> 'b -> 'a list -> 'b`
`foldl f b [x1, x2, ..., xn]` liefert `f(xn,...,f(x2, f(x1, b))...)` oder `b`, wenn die Liste leer ist, zurück.

g) `exists : ('a -> bool) -> 'a list -> bool`
`exists f l` wendet `f` auf Elemente `x` aus `l` von links nach rechts solange an, bis `f x` den Wert `true` zurückliefert; liefert den Wert `true` zurück, falls so ein `x` existiert, sonst `false`.

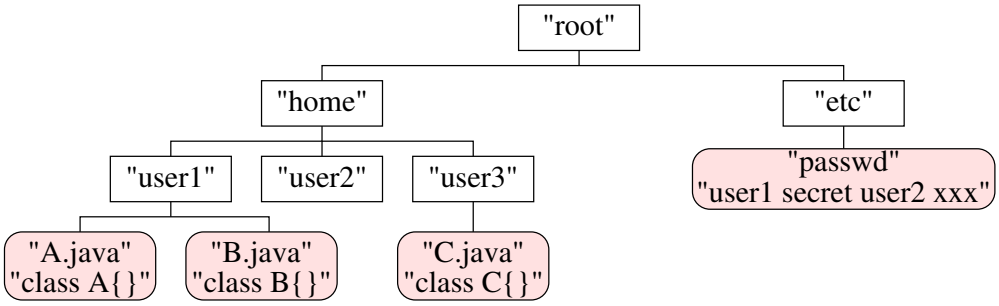
h) `all : ('a -> bool) -> 'a list -> bool`
`all f l` wendet `f` auf Elemente `x` aus `l` von links nach rechts solange an, bis `f x` den Wert `false` zurückliefert; liefert den Wert `false` zurück, falls so ein `x` existiert, sonst `true`.

i) `tabulate : int * (int -> 'a) -> 'a list`
`tabulate (n, f)` liefert die Liste `[f(0), f(1), ..., f(n-1)]` zurück.

Aufgabe 2:

a) Definiere einen SML-Typ `node`, mit dem sich Dateien und Verzeichnisse als Knoten eines Filesystem-Baums darstellen lassen. Eine Datei besteht aus einem Namen vom Typ `string` und einem Inhalt, der (der Einfachheit halber) auch vom Typ `string` ist. Ein Verzeichnis besteht aus einem Namen vom Typ `string` und aus einer (möglicherweise leeren) Liste von Unterverzeichnissen und Dateien.

Ein Beispiel-Filesystem ist unten abgebildet. Verzeichnisse sind durch Rechtecke und Dateien durch abgerundete graue Rechtecke dargestellt:



b) Schreibe eine Funktion `addNode : node -> node -> node`, die ein Verzeichnis d_1 und einen Knoten n als Argumente bekommt und das Verzeichnis d_2 zurückliefert, in dem n zu dem Inhalt von d_1 hinzugefügt wird.

c) Implementiere eine Funktion `changeDir : node -> string list -> node`, die als Argumente ein Verzeichnis d und einen Pfad *path* (als Liste von `string`s) erhält. Wenn `path` zu einem Unterverzeichnis d_1 aus dem Verzeichnis d führt, soll d_1 zurückgeliefert werden.