Andreas Podelski
Fachbereich 14 – Informatik
Universität des Saarlandes,
Max-Planck-Institut für Informatik

Programmiersprachen (WS 1996/97)

7. Nov 1996

Übungsblatt 3

Abdelwaheb Ayari
Hubert Baumeister

Aufgabe 1: (2 Punkte)
Die Fibonacci Funktion fib: int $\rightarrow$ int ist wie folgt definiert

```
fun fib 0 = 0
  | fib 1 = 1
  | fib i = fib (i-2) + fib (i-1);
```

Eine effizientere Fibonacci Funktion läßt sich implementieren, wenn man die Idee der folgenden C Funktion zugrundelegt:

```
int fib (int n) {
   if (n == 0) return 0;
   int i = n, prev = 0, curr = 1, h;
   while (i > 1) {
      h = curr; curr = prev + curr; prev = h;
      i--;
   };
   return curr;
}
```

Schreibe, ausgehend von der C Funktion, in ML eine effizientere Fibonacci Funktion. Benutze dabei eine Funktion ifib: int $*$ int $*$ int $\rightarrow$ int $*$ int $*$ int.

Aufgabe 2: (2 Punkte)
Schreibe eine Funktion while_loop: $(\alpha \rightarrow \text{bool}) * (\alpha \rightarrow \alpha) \rightarrow \alpha \rightarrow \alpha$ die zu Funktionen $b\colon \alpha \rightarrow$ bool und $f\colon \alpha \rightarrow \alpha$ eine Funktion while_loop(b, f) liefert, die der folgenden (pseudo) C Funktion entspricht:

```
'a schleife ('a x) {
   'a s = x;
   while (b(s)) {
      s = f(s);
   };
   return s;
}
```

Gib Funktionen b und f an, so daß ifib = while_loop(b, f) ist.

Aufgabe 3: (3 Punkte)
Implementiere in ML für Binärbäume mit Integer-Schlüsseln

```
datatype inttree = leaf | node of int * inttree * inttree;
```

die folgenden Funktionen:

1. `insert : inttree * int -> inttree` // Einfügen eines Schlüssels in den Binärbaum.
2. `top : inttree -> int` // Schlüssel der Wurzel.
3. `remove : inttree -> inttree` // Lösche den Schlüssel des Wurzelknotens.

Stelle sicher daß der Schlüssel jedes Sohnes kleiner oder gleich ist als der Schlüssel des Vaters. Implementiere ein Sortierverfahren mit Hilfe der Funktionen `insert`,`top` und `remove`.

Aufgabe 4: (3 Punkte)
Implementiere einen Datentyp für arithmetische Ausdrücke, die aus Variablen, ganzzahlige Konstanten, Summen und Produkten aufgebaut sind. Beispiele für arithmetische Ausdrücke sind:

```
val x = var("x")
val y = var("y")
val z = var("z")
val e = prod(x, sum (x,y));
val zero = const(0)
val one = const(1);
```

Schreibe eine Funktion `simplify`, die einen arithmetischen Ausdruck bzgl. der Regeln $e * 0 = 0$, $e * 1 = e$, $e + 0 = e$ und $e_1 * (e_2 + e_3) = e_1 * e_2 + e_1 * e_3$ vereinfacht.

Die Ableitung eines Ausdrucks nach einer Variablen erfolgt nach den üblichen Regeln:

- die Ableitung einer Konstanten c nach x ist 0,
- die Ableitung von y nach x ist 1, falls $x = y$ und 0 sonst,
- die Ableitung einer Summe $e_1 + e_n$ nach x ist die Summe von e_1 und e_2 abgeleitet nach x und
- die Ableitung eines Produkts $e_1 * e_n$ nach x ist die Summe des Produkts der Ableitung von e_1 nach x mit e_2 und des Produkts der Ableitung von e_2 nach x mit e_1.

Schreibe eine Funktion d, die eine Variable x und einen Ausdruck e als Argument nimmt und die Ableitung von e nach x bildet. Der Ergebnisausdruck soll vollständig vereinfacht sein. Beispiel:

```
d(x,e) = sum (sum (var "x",var "y"),var "x")
```

Tip: Mit der Eingabe von `System.Control.Print.printDepth := 100;` kann man die Ausgabe von komplexen Strukturen in ML verbessern.