DEPENDABLE SYSTEMS AND SOFTWARE

Fachrichtung 6.2 — Informatik
Prof. Dr.-Ing. Holger Hermanns
Dipl.-Inform. Lijun Zhang

Übungsblatt 6 (Programmierung I)

Lesen Sie im Skript: Kapitel 6

Aufgabe 6.1: (Variantennummern)
Deklarieren Sie eine Prozedur *variant* : *shape* → *int*, die die Variantennummer eines geometrischen Objekts liefert, Beispielsweise soll $variant(Square\ 3.0) = 2$ gelten.

Aufgabe 6.2: (Variablen in einem Ausdruck)
Für den Datentype *exp* aus Kapitel 6.4, deklarieren Sie eine Prozedur *vars* : *exp* → *var list*, die zu einem Ausdruck eine Liste liefert, die die in dem Ausdruck vorkommenden Variablen enthält. Orientieren Sie sich an der Prozedur *subexps*.

Aufgabe 6.3: (Variablen zählen)
Deklarieren Sie eine Prozedur *count* : *var* → *exp* → *int*, die zählt, wie oft eine Variable in einem Ausdruck auftritt. Beispielsweise tritt x in $x + x$ zweimal auf.

Aufgabe 6.4: (Teilausdrücke)
Deklarieren Sie eine Prozedur *check* : *exp* → *exp* → *bool*, die für zwei Ausdrücke e und e' testet, ob e ein Teilausdruck von e' ist.

Aufgabe 6.5: (Instantiierungen)
Schreiben Sie eine Prozedur *instantiate* : *env* → *exp* → *exp*, die zu einer Umgebung V und einem Ausdruck e den Ausdruck liefert, den man aus e erhält, indem man die in e vorkommenden Variablen gemäß V durch Konstanten ersetzt. Beispielsweise soll für die oben deklarierte Umgebung *env* und den Ausdruck $A(V1, V2)$ der Ausdruck $A(C5, C3)$ geliefert werden. Orientieren Sie sich an der Prozedur *eval*.

Aufgabe 6.6: (Symbolisches Differenzieren)
Sie sollen eine Prozedur schreiben, die Ausdrücke nach der Variable x ableitet. Hier ist ein Beispiel:

$$(x^3 + 3x^2 + x + 2)' = 3x^2 + 6x + 1$$

Ausdrücke sollen gemäß des folgenden Typs dargestellt werden:

```
datatype exp = C of int           c
             | X                  x
             | A of exp * exp     u + v
             | M of exp * exp     u * v
             | P of exp * int     u^n
```

(a) Schreiben Sie eine Deklaration, die den Bezeichner u an die Darstellung des Ausdrucks $(x^3 + 3x^2 + x + 2)$ bindet. Der Operator + soll nach links gruppieren.

(b) Schreiben sie eine Prozedur *derive* : *exp* → *exp*, die die Ableitung eines Ausdrucks gemäß den folgenden Regeln berechnet:

$$c' = 0$$
$$x' = 1$$
$$(u + v)' = u' + v'$$
$$(u \cdot v)' = u' \cdot v + u \cdot v'$$
$$(u^n)' = n \cdot u^{n-1} \cdot u'$$

Die Ableitung darf vereinfachbare Teilausdrücke enthalten(z.B. $0 \cdot u$).

Aufgabe 6.7: (Konstruktordarstellung der natürlichen Zahlen)
In dieser Aufgabe stellen wir die natürlichen Zahlen mit den Werten des Typs

```
datatype nat = O | S of nat
```

wie folgt dar: $0 \mapsto O$, $1 \mapsto SO$, $2 \mapsto S\,(SO)$, $3 \mapsto S\,(S\,(SO))$, und so weiter.

a) Deklarieren Sie eine Prozedur $rep : int \to nat$, die die Darstellung einer natürlichen Zahl liefert.
b) Deklarieren Sie eine Prozedur $num : nat \to int$, die zu einer Darstellung die dargestellte Zahl liefert.
c) Deklarieren Sie für *nat* eine kaskadierte Prozedur *add*, die der Operation + für natürliche Zahlen entspricht. Verwenden Sie dabei keine Operationen für *int*.

Aufgabe 6.8: (Konstruktordarstellung der ganzen Zahlen)
In dieser Aufgabe stellen wir die ganzen Zahlen mit den Werten des Typs

```
datatype integer = N of nat | P of nat
```

wie folgt dar: Der Typ *nat* sei wie in Aufgabe 6.7 definiert.

(a) Deklarieren Sie eine Prozedur $rep' : int \to integer$, die die Darstellung einer ganzen Zahl liefert. Verwenden Sie die Prozedur *rep* für *nat*.

(b) Deklarieren Sie eine Prozedur $num' : integer \to int$, die zu einer Darstellung die dargestellte Zahl liefert. Verwenden Sie die Prozedur *num* für *nat*.

(c) Deklarieren Sie für *integer* eine kaskadierte Prozedur add', die der Addition für ganze Zahlen entspricht. Verwenden Sie dabei die Prozedur *add* für *nat*. Sie benötigen 8 Regeln für add'. Wenn Sie die Kommutativität der Addition ausnützen und die letzte Regel die Argumente vertauschen lassen, können Sie mit 6 Regeln auskommen.

Aufgabe 6.9: (Test auf Darstellbarkeit)
Schreiben Sie eine Prozedur $test : int \to bool$, die testet, ob das Quadrat einer ganzen Zahl im darstellbaren Zahlbereich liegt.

Aufgabe 6.10: (Sequenzialisierung mit Abstraktion und Applikation)
Führen Sie zweistellige Sequenzialisierungen $(e_1; e_2)$ auf Abstraktionen und Applikationen zurück.

Aufgabe 6.11: (Variante von testDouble)
Schreiben Sie die Prozedur *testDouble* so um, dass die Ausnahme *Double* und die Hilfsprozedur *mask* mithilfe eines Let-Ausdrucks lokal deklariert werden.

```
exception Double

fun mask compare p = case compare p of EQUAL => raise Double | v => v

fun testDouble compare xs = (Listsort.sort (mask compare) xs ; false) handle Double => true
```

Aufgabe 6.12: (Laufzeitbetrachtung)
Die Prozedur *testDouble* aus § 6.5.5 testet auch sehr lange Listen schnell auf Mehrfachauftreten. Schreiben Sie einen Test auf Mehrfachauftreten, der ohne Sortieren arbeitet, und überzeugen Sie sich mit der Liste [1, ..., 10000] davon, dass *testDouble* sehr viel schneller ist. Daran ändert sich auch nichts, wenn Sie statt *Listsort.sort* eine selbstgeschriebene Sortierprozedur verwenden, die durch Mischen sortiert.

Aufgabe 6.13: (Letztes Element einer Liste)
Schreiben Sie eine Prozedur $last : \alpha\ list \to \alpha\ option$, die das letzte Element einer Liste liefert.

Aufgabe 6.14: (Liste der ersten n Primzahlen)
Deklarieren Sie eine Prozedur $primelist : int \to int\ list$, die zu $n \geq 0$ die Liste der ersten n Primzahlen in aufsteigender Ordnung liefert. Verwenden Sie die Prozedur *nextprime*. Berechnen Sie die Liste der ersten n Primzahlen ab einer gegebenen Primzahl.
Hinweis: Sie benötigen ein Akkumulatorargument.