10. Übungsblatt zu Programmierung WS 2002 / 03

Prof. Dr. Gert Smolka und Dipl.-Inform. Thorsten Brunklaus
`www.ps.uni-sb.de/courses/prog-ws02/`

Abgabe: Montag, 13. Januar 2003

Schicken Sie Ihre Lösungen per E-Mail an Ihren Übungsgruppenleiter. Geben Sie in der E-Mail zuerst ihren Name und die Nummer Ihrer Übungsgruppe an. Bitte reichen Sie alle Lösungen mit nur einer E-Mail ein.

Aufgabe 10.1: Signaturen (4+4+4)

(a) Betrachten Sie die Strukturdeklaration in Abbildung 9.2 (Skript).Welche Signatur wird dem Bezeichner `ISet` zugeordnet, wenn man den Signaturconstaint `:> ISET` weglässt?

(b) Betrachten Sie die Funktordeklaration in Abbildung 9.5 (Skript).Welche Signatur ordnet die Deklaration

```
structure StringSet =
  Set(type elem = string
      val compare = String.compare)
```

dem Bezeichner `StringSet` zu?

(c) Warum ist die Deklaration

```
structure S :> sig eqtype t end =
  struct type t = int → int end
```

unzulässig?

Aufgabe 10.2: Rot-Schwarz-Bäume (4+4+6+6+4)

(a) Schreiben Sie eine Prozedur

```
listA : ctree → int list
```

die zu einem Rot-Schwarz-Baum eine aufsteigend sortierte Liste liefert, die die Elemente der dargestellten Menge enthält. Die Laufzeit der Prozedur soll linear bezüglich der Größe der dargestellten Mengen sein. Verwenden Sie eine Hilfsprozedur mit Akkumulatorargument.

(b) Schreiben Sie eine Prozedur

```
listD : ctree → int list
```

die zu einem Rot-Schwarz-Baum eine absteigend sortierte Liste liefert, die die Elemente der dargestellten Menge enthält. Die Laufzeit der Prozedur soll linear bezüglich der Größe der dargestellten Mengen sein. Verwenden Sie eine Hilfsprozedur mit Akkumulatorargument.

(c) Schreiben Sie eine Prozedur

```
sb : ctree → bool
```

die testet, ob ein C-Baum (als S-Baum betrachtet) ein Suchbaum ist.

(d) Schreiben Sie eine Prozedur

```
rb : ctree → int (* NotRB *)
```

die die S-Höhe eines RB-Baums liefert. Die Prozedur soll zusätzlich testen, ob es sich bei ihrem Argument um einen RB-Baum handelt. Wenn dies nicht der Fall ist, soll die Ausnahme `NotRB` geworfen werden. Gehen Sie dabei ähnlich vor wie beim Test auf Balanciertheit in Abschnitt 5.10 (Skript). Verwenden Sie eine Hilfsprozedur `forward`. Verwenden Sie außerdem eine Hilfsprozedur

```
black : ctree → unit (* NotRB *)
```

die für einen C-Baum die Ausnahme `NotRB` wirft, wenn seine Wurzel rot ist.

(e) Schreiben Sie eine Prozedur

```
redblack : ctree → bool
```

die testet, ob ein C-Baum ein Rot-Schwarz-Baum ist.

Aufgabe 10.3: Funktor für effiziente endliche Mengen (10) Schreiben Sie analog zu Abschnitt 9.4 (Skript) einen Funktor `Set`, der endliche Mengen mit Rot-Schwarz-Bäumen implementiert.

Aufgabe 10.4: Funktor für effiziente endliche Funktionen (10) Schreiben Sie analog zu Aufgabe 9.2 einen Funktor `Map`, der endliche Funktionen mit Rot-Schwarz-Bäumen implementiert. Verwenden Sie dabei einen Konstruktortyp für C-Bäume, der Ihnen erlaubt, die in Abbildung 10.3 (Skript) gezeigten Rotationsprozeduren ohne Änderung zu übernehmen. Verwenden Sie eine Hilfprozedur `compare'`, die Ihnen erlaubt, die Prozedur `insert'` in Abbildung 10.3 (Skript) zu übernehmen, nachdem Sie `Int.compare` durch `compare'` ersetzt haben.

Aufgabe 10.5: Große natürliche Zahlen ($44 = 6 + 6 + 0 + 10 + 4 + 10 + 8$) Implementieren Sie eine Typabstraktion für große natürliche Zahlen gemäß der Signatur

```
eqtype bignat

val fromString : string → bignat
val toString   : bignat → string
val add        : bignat * bignat → bignat
val sub        : bignat * bignat → bignat        (* Domain *)
val less       : bignat * bignat → bool
val mul        : bignat * bignat → bignat
```

Gehen Sie dabei so wie in Abschnitt 10.2 (Skript) und nachfolgend beschrieben vor.

Im Folgenden benutzen wir das Typsynonym

```
type digit = int
```

für die Menge der Ziffern bezüglich der Basis B (also die Zahlen 0 bis $B-1$).

Da `bignat` als Typ mit Gleichheit sichtbar ist, dürfen keine führenden Nullen eingeführt werden. Dabei ist eine Hilfsprozedur

```
adj : digit * bignat → bignat
```

nützlich, die zu (d, x) die Zahl $d + xB$ berechnet.

Verwenden Sie eine Hilfsprozedur

```
fromInt : int → bignat
```

Implementieren Sie die Operation `mul` mit einer Hilfsprozedur

```
mul' : bignat * digit * digit → bignat
```

die zu (x, b, c) die Zahl $xb + c$ berechnet. Die Prozedur `mul` soll gemäß der Gleichung

$$x(y + y'B) = xy + (xy')B$$

realisiert werden.

Implementieren Sie `toString` ohne Division indem Sie ausnützen, dass Sie mit der Basis 10000 arbeiten. Alle anderen Operationen sollen jedoch so realisiert werden, dass sie für beliebige Basen arbeiten.

Realiseren Sie Ihre Implementierung großer natürlicher Zahlen durch eine Komponente `Bignat`. Schreiben Sie mithilfe von `Bignat` ein direkt ausführbares Programm, dass zu einem Startargument $n \in \mathbb{N}$ die Fakultät $n!$ ausgibt. Berechnen Sie damit 1000!.