10. Übungsblatt zu Programmierung

Prof. Gert Smolka, Thorsten Brunklaus
www.ps.uni-sb.de/courses/prog-ws00/

Abgabe: 26. Januar 2001 vor der Vorlesung (per Email)

Allgemeine Hinweise: Die Übungsblätter sollen in Zweiergruppen bearbeitet werden. Die Lösungen schicken Sie bitte bis Freitag vor der Vorlesung per Email an Ihren Übungsgruppenleiter. Jede Gruppe soll nur eine Lösung einreichen, versehen mit den Namen und den Matrikelnummern der Gruppenmitglieder.

Aufgabe 10.1: Signaturen (2+2+2)

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

(b) Betrachten Sie die Funktordeklaration in Abbildung 9.5.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 (2+2+3+3)

(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.

(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.

(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. Wenn das Argument kein RB-Baum ist, soll die Ausnahme `NotRB` geworfen werden. Gehen Sie dabei ähnlich vor wie beim schnellen Test auf Balanciertheit (siehe Abschnitt 6.8).Verwenden Sie außerdem eine Hilfsprozedur

```
black : ctree -> unit (* NotRB *)
```

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

Aufgabe 10.3: Funktor für effiziente endliche Mengen (8) Schreiben Sie analog zu Abschnitt 9.4 einen Funktor `Set`, der endliche Mengen mit Rot-Schwarz-Bäumen implementiert.

Aufgabe 10.4: Funktor für effiziente endliche Funktionen (6) 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 gezeigten Rotationsprozeduren ohne Änderung zu übernehmen. Verwenden Sie zudem eine Hilfprozedur `compare'`, die Ihnen erlaubt, die Prozedur `insert'` in Abbildung 10.3 zu übernehmen, nachdem Sie `Int.compare` durch `compare'` ersetzt haben.

Aufgabe 10.5: Große Natürliche Zahlen (3+3+2+0+4+4+4) Implementieren Sie große natürliche Zahlen gemäß der Signatur

```
eqtype bignat

val fromString : string -> bignat
val toString   : bignat -> string
val less       : bignat * bignat -> bool
val add        : bignat * bignat -> bignat
val sub        : bignat * bignat -> bignat  (* Domain *)
val mul        : bignat * bignat -> bignat
val divmod     : bignat * bignat -> bignat * bignat  (* Div *)
```

Verwenden Sie die Darstellung aus Abschnitt 10.2. Da `bignat` als Typ mit Gleichheit sichtbar ist, dürfen keine führenden Nullen eingeführt werden (Vorsicht bei der Subtraktion!). Die Subtraktionsoperation `sub` soll die vordefinierte Ausnahme `Domain` werfen, wenn das zweite Argument größer ist als das erste. Die Divisionsoperation `divmod` soll zu $(x, y) \in \mathbb{N} \times \mathbb{N}^+$ das Paar $(\lfloor x/y \rfloor,\ x - \lfloor x/y \rfloor\, y)$ liefern. Für $(x, 0)$ soll `divmod` die Ausnahme `Div` werfen.