7. Übungsblatt „Programmiersprachen“ WS 97/98

Leif Kornstaedt, Prof. Gert Smolka

Ausgabe: 11. Dezember 1997 **Abgabe:** 22. Dezember 1997

VII.1: Darstellung von Mengen in SML (6 Punkte)

Gegeben sei folgende Datentypdefinition zur Darstellung von Mengen:

```
type ''a set = ''a list
```

Das doppelte Quote vor der Typvariablen besagt, daß für den Elementtyp der Test auf Gleichheit definiert sein muß.

Bei dieser Darstellung kann nicht vermieden werden, daß Mengen unter Umständen Elemente mehrfach enthalten. Implementieren Sie folgende Konstante und Operationen:

```
emptyset: ''a set                       (* Leere Menge *)
adjoin: ''a * ''a set -> ''a set        (* Element hinzufügen *)
card: ''a set -> int                    (* Kardinalität *)
member: ''a * ''a set -> bool           (* Ist-Element-von *)
subset: ''a set * ''a set -> bool       (* Teilmengenrelation *)
union: ''a set * ''a set -> ''a set     (* Mengenvereinigung *)
```

Hinweis: Für die Implementierung der Kardinalitätsfunktion müssen Sie auf doppelt enthaltene Elemente achten – diese dürfen jeweils nur einfach gezählt werden.

VII.2: Abstrakte Datentypen (5 + 3 Punkte)

a. Implementieren Sie die Mengen aus Aufgabe 1 mit den gegebenen Operationen als abstrakten Datentyp unter Verwendung von **`abstype`**. Stellen Sie hierbei sicher, daß jedes Element nur einfach in der Menge enthalten ist.

   Hinweis: Sie sind in dieser Aufgabe nicht mehr an die Darstellung durch Listen gebunden, wenn Sie sie auch weiterhin verwenden dürfen.

b. Definieren Sie eine **`structure`** `Set`, die den obigen abstrakten Datentyp und seine Operationen bereitstellt.

VII.3: Funktoren in SML $(5 + 6 + 5 + 3$ Punkte$)$

a. Gegeben sei folgende Signatur:

```
signature SET0 =
    sig
        eqtype element
        type set
        val emptyset: set
        val adjoin: element * set -> set
        val foldl: (element * 'a -> 'a) -> 'a -> set -> 'a
    end
```

Implementieren Sie eine **structure** `Set0` mit der Signatur `SET0` zur Darstellung von Mengen. Wählen Sie `int` als Elementtyp. Der Typ `set` soll abstrakt implementiert werden.

Die Operation `foldl` ist für eine Funktion f, einen Wert e und eine (eventuell leere) Menge $\{x_1, \ldots, x_n\}$ wie folgt definiert:

$$\texttt{foldl}\ f\ e\ \{x_1, \ldots, x_n\} = f\ (x_n, f\ (\ldots, f\ (x_2, f\ (x_1, e))\ldots))$$

Dabei ist die Reihenfolge der x_i beliebig; alle x_i müssen jedoch paarweise verschieden sein. Bemerken Sie, daß für `foldl` ausnahmsweise (in Anlehnung an den Typ der SML-Standardfunktion `foldl`) *currying* verwendet wird.

b. Betrachten Sie nun folgende Signatur `SET1`, die die Signatur `SET0` um vier Operationen erweitert:

```
signature SET1 =
    sig
        eqtype element
        type set
        val emptyset: set
        val adjoin: element * set -> set
        val foldl: (element * 'a -> 'a) -> 'a -> set -> 'a
        val card: set -> int
        val member: element * set -> bool
        val subset: set * set -> bool
        val union: set * set -> set
    end
```

Schreiben Sie einen Funktor wie folgt:

```
functor Set1 (Set0: SET0): SET1 =
    struct
        ...
    end
```

Das bedeutet, daß Sie die relativ zur Signatur `SET0` hinzugekommenen Operationen auf `foldl` zurückführen müssen.

Hinweis: Sofern Sie Ihre Mengen als Listen darstellen, sollten Sie für die Implementierung von `member` und `subset` Ausnahmebehandlung verwenden, um das Durchlaufen der Restliste zu vermeiden, sobald das Ergebnis sowieso feststeht.

c. Die Mengen aus der vorigen Teilaufgabe sollen nun mit einer Totalordnung auf den Elementen versehen werden. Betrachten Sie hierzu folgende Datentypdefinition und Signatur:

```
datatype compareResult = Less | Equal | Greater

signature TOTALORDER =
    sig
        eqtype t
        val compare: t * t -> compareResult
    end
```

Schreiben Sie einen Funktor wie folgt, für dessen Operation `foldl` gilt, daß die Mengenelemente x_i immer in (von innen nach außen) aufsteigender Reihenfolge verknüpft werden:

```
functor OrderedSet0 (TotalOrder: TOTALORDER): SET0 =
    struct
        ...
    end
```

d. Schreiben Sie eine **`structure`** `IntOrder` mit der Signatur `TOTALORDER`, die die in der Mathematik übliche Totalordnung auf den ganzen Zahlen implementiert. Definieren Sie dann mithilfe von `OrderedSet0` eine **`structure`** `IntSet0` mit den grundlegenden Mengenoperationen, daraufhin unter Verwendung von `Set1` eine **`structure`** `IntSet1`, die auch die anderen Mengenoperationen zur Verfügung stellt.

Hinweis: Wenn Sie es richtig machen, ist für die Definition von `IntSet0` und `IntSet1` jeweils nur eine einzige Zeile nötig.