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

Programmiersprachen (WS 1996/97)

12. Dezember 1996

Übungsblatt 8

Abdelwaheb Ayari
Hubert Baumeister

Aufgabe 1: (5 Punkte)

1. Definiere eine ML-Struktur `SetAsList : SET`, die den Datentyp Menge mit Hilfe einfacher Listen implementiert.

```
signature SET =
  sig
    type ''a T
    val empty: ''a T
    val isEmpty: ''a T -> bool
    val mem : ''a * ''a T -> bool
    val insert : ''a * ''a T -> ''a T
    val delete : ''a * ''a T -> ''a T
    val intersect : ''a T * ''a T -> ''a T
    val union : ''a T * ''a T -> ''a T
  end;
```

2. Gegeben seien zwei `SET`-Algebren A und B. Die Algebra A ist durch die Struktur `SetAsList` gegeben mit $A_{\alpha\ \texttt{SET.T}} = \alpha\ \texttt{list}$ und die Algebra B ist die Algebra der Mengen mit Elementen aus $A_\alpha = B_\alpha$, das heißt $B_{\alpha\ \texttt{SET.T}} = \{\ S \mid S \subseteq B_\alpha\ \}$ mit den Operationen $\texttt{empty}_B = \emptyset$, $\texttt{mem}_B = \in$, etc. Definiere eine Abstraktionsfunktion $a : A_{\alpha\ \texttt{SET.T}} \to B_{\alpha\ \texttt{SET.T}}$ und eine Repräsentationsinvariante $I \subseteq A_{\alpha\ \texttt{SET.T}}$.

3. Zeige, daß die `SET`-Algebra A eine Implementierung der `SET`-Algebra B bezüglich der Sorte α `SET.T` ist.

Aufgabe 2: (5 Punkte)

1. Definiere eine Struktur `Stream : STREAM`, die die unendlichen Ströme aus der Vorlesung implementiert.

```
signature STREAM =
   sig
     type 'a T
     exception EmptyStream
     val Nil : 'a T                      val Cons : 'a * (unit -> 'a T) -> 'a T
     val isNil : 'a T -> bool            val tail : 'a T -> 'a T
     val head : 'a T -> 'a               val drop : int * 'a T -> 'a T
     val take : int * 'a T -> 'a list
     val foldl : ('a *(unit -> 'b) -> 'b) -> 'b -> 'a T -> 'b
   end;
```

2. Gegeben seien Signaturen

```
signature ANY =              signature ORDER =
  sig                          sig
    type T                       type T
  end;                           val less : T * T -> bool
                               end;
```

Definiere eine Signatur `STORE` und implementiere Funktoren

`functor StackFUN(X:ANY):STORE`, `functor QueueFUN(X:ANY):STORE` und
`functor PQueueFUN(X:ORDER):STORE`,

die zu einer Struktur der Signatur `ANY` bzw. `ORDER` einen Stack, eine Queue bzw. eine Priority Queue liefern, die die Operationen `isEmpty`, `empty`, `add`, `addAll`, `element`, `remove` enthalten sollen. Dabei nimmt `addAll` eine Liste von Elementen und wendet die `add` Operation auf jedes einzelne Element der Liste an, das heißt `addAll(h::t,l)` = `add(h,addAll(t,l))`. Für einen Store s liefert die Funktion `element`(s) das Element, das bei `remove`(s) aus s entfernt wird.

3. Die Signatur `PROBLEM` enthalte die Komponenten `type T`, `next: T -> T list`, `pred: T -> bool` und `less: T*T -> bool`. Die Funktion `less` vergleicht zwei Konfigurationen k und k'; ist `less`(k, k') = `true`, dann wird k vor k' bei der Bestensuche expandiert. Gegeben sei die Signatur

```
signature SEARCH =
  sig
    structure Stream:STREAM
    type T
    val search : T -> T Stream.T
 end;
```

Definiere einen Funktor `SearchFUN`, der eine Struktur der Signatur `PROBLEM`, eine Struktur der Signatur `STORE` und eine Implementierung der Signatur `STREAM` nimmt und eine Struktur der Signatur `SEARCH` liefert. `SearchFUN` angewandt auf eine Beschreibung eines Suchproblems und entweder Stack, Queue oder Priority Queue liefert die Suchverfahren Tiefensuche, Breitensuche oder Bestensuche als Ergebnis.

4. Definiere eine Struktur der Signatur `PROBLEM` mit `next`$(i) = [2{*}i, 2{*}i{+}1]$ und `pred`$(i) = i > 3$. Lasse die ersten 10 Lösungen für Tiefensuche, Breitensuche und Bestensuche berechnen. Bei der Bestensuche verwende einmal `less`$(i, j) = i < j$ und einmal `less`$(i, j) = i > j$.