3. Übungsblatt zu Programmierung

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

Abgabe: 17. November 2000 in der Vorlesungspause (auf Papier)

Allgemeine Hinweise: Die Übungsblätter sollen in Zweiergruppen bearbeitet werden. Jede Gruppe soll nur eine Lösung einreichen, versehen mit den Namen und den Matrikelnummern der Gruppenmitglieder.

Spezieller Hinweis: Für die meisten Aufgaben sind gezielte Interpreter-Experimente sehr hilfreich.

Aufgabe 3.1: Höherstufige Prozeduren (2+2+2+2)

(a) Deklarieren Sie eine Prozedur

```
prod : (int -> int) * int -> int
```

die die folgenden Gleichungen erfüllt:

```
prod(f,0) = 1
prod(f,n) = f n * prod(f,n-1)    falls n > 0
```

Tun Sie dies mit einer Prozedurdeklaration.

(b) Deklarieren Sie `prod` mit einer rekursiven Wertdeklaration.

(c) Deklarieren Sie eine kaskadierte Variante von `prod` mit einer Prozedurdeklaration.

(d) Deklarieren Sie eine kaskadierte Variante von `prod` mit einer rekursiven Wertdeklaration.

Aufgabe 3.2: Kartesische und kaskadierte Prozeduren (3+3) Deklarieren Sie zwei Prozeduren

```
kas : ('a * 'b -> 'c) -> 'a -> 'b -> 'c

kar : ('a -> 'b -> 'c) -> 'a * 'b -> 'c
```

für die gilt:

```
kar (kas op+) (3,4)
7 : int


kas (kar (kas op+)) 3 4
7 : int
```

Dabei soll `kas` zu einer kartesischen Prozedur eine gleichwertige kaskadierte Prozedur und `kar` zu einer kaskadierten Prozedur eine gleichwertige kartesische Prozedur liefern.

Aufgabe 3.3: Monomorphe Typsynthese (14) Geben Sie Wertdeklarationen an, die die folgenden statischen Bindungen einführen:

```
val a : int * unit * bool
val b : unit * (int * unit) * (real * unit)
val c : int -> int
val d : int * bool -> int
val e : int -> real
val f : int -> real -> real
val g : (int -> int) -> bool
```

Dabei dürfen Sie die folgenden Dinge *nicht* verwenden:

- Typconstraints für Argumentvariablen.
- Prozeduranwendungen.
- Operationen (zum Beispiel `+`, `-`, `<` und `=`).
- Standardstrukturen.

Hinweis: Für einige der Bindungen ist die Verwendung eines Konditionals essentiell. Die Typregel für das Konditional verlangt, dass die Konsequenz und die Alternative den gleichen Typ haben. Für einige Bindungen benötigen Sie zusätzlich Let-Ausdrücke

```
let val x = a1 in a2 end
```

bei denen `x` nicht in a_2 vorkommt.

Aufgabe 3.4: Polymorphe Typsynthese (2+2+2+2) Geben Sie für jedes der folgenden Typschemata einen polymorphen Ausdruck an, der genau die durch das Typschema beschriebenen Typen hat. Die Ausdrücke dürfen keine Typconstraints und keine freien Bezeichnerauftreten enthalten.

(a) `∀ 'a ('a -> 'a)`

(b) `∀ ''a (''a -> ''a)`

(c) `∀ ''a 'b (''a -> 'b -> 'b * ''a)`

(d) `∀ 'a 'b 'c (('a -> 'b -> 'c) -> 'a -> 'b -> 'c)`

Denken Sie daran, dass Typvariablen mit zwei Hochkommas nur mit Typen mit Gleichheit instanziiert werden können. Für einige Beispiele benötigen Sie den Gleicheitstest.

Aufgabe 3.5: Bedeutungsgleiche Ausdrücke (3+2) Gegeben sei der folgende Ausdruck:

```
fn x => fn y => (fn x => (fn y => (fn x => y) x) y) x
```

(a) Geben Sie einen möglichst einfachen bedeutungsgleichen Ausdruck an.

(b) Geben Sie ein Typschema an, dass alle Typen des Ausdrucks beschreibt.

Aufgabe 3.6: Lexikalische Bindungen (1+1+2) Betrachten Sie den Ausdruck

```
(fn x => (fn y => (fn x => y) x) y) x
```

(a) Stellen Sie die lexikalischen Bindungen des Ausdrucks durch Überstreichen und Indizieren von Bezeichnerauftreten dar.

(b) Welche Bezeichner kommen in dem Ausdruck frei vor?

(c) Geben Sie einen bedeutungsgleichen bereinigten Ausdruck an.

Aufgabe 3.7: Lexikalische Bindungen (5) Stellen Sie die lexikalischen Bindungen des folgenden Ausdrucks durch Überstreichen und Indizieren von Bezeichnerauftreten dar.

```
let
    val f = fn x => fn x => f x y
    val x = 2*x
    val x =(x,y,f)
    val rec g = fn n => if n<2 then 1 else n*g(n-1)
    fun f f = f x
in
    fn x => f x
end
```

Welche Bezeichner kommen in dem Ausdruck frei vor?