3. 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, 11. November 2002

Alle Aufgaben sind auf Papier im Briefkasten Nr. 15 im Durchgang zwischen den Gebäuden 36 und 45 abzugeben. Ihre Abgaben können nur berücksichtigt werden, wenn Sie jeweils Ihren Namen und **die Nummer Ihrer Übungsgruppe angeben**.

Aufgabe 3.1: Kartesische und kaskadierte Prozeduren $(6 + 6)$ 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 3.2: Monomorphe Typsynthese $(21 = 7*3)$ Geben Sie Deklarationen 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 =` a_1 `in` a_2 `end`

bei denen `x` nicht in a_2 vorkommt.

Aufgabe 3.3: Polymorphe Typsynthese ($16 = 4 * 4$) 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 Gleichheitstest.

Aufgabe 3.4: Bedeutungsgleiche Ausdrücke ($6 + 4$) 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 3.5: Lexikalische Bindungen ($2 + 2 + 4$) 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 3.6: Lexikalische Bindungen (8) 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?

Aufgabe 3.7: Primzahlen ($25 = 5 + 5 + 5 + 8 + 2$**)** Sie sollen eine Prozedur schreiben, die zu $n \geq 1$ die n-te Primzahl liefert. Diese Aufgabe ist nicht einfach, obwohl das gesuchte Programm nur aus wenigen Zeilen besteht. Deswegen helfen wir Ihnen mit einer Zerlegung des Problems in einfachere Teilprobleme. Zunächst sollten Sie sich mit den folgenden Tatsachen über Primzahlen vertraut machen:

- Eine natürliche Zahl $n \geq 2$ heißt Primzahl, wenn sie nur durch 1 und sich selbst teilbar ist.
- Die ersten 10 Primzahlen sind 2, 3, 5, 7, 11, 13, 17, 19, 23, 29.
- Es gibt unendlich viele Primzahlen.
- Eine natürliche Zahl $n \geq 2$ ist genau dann eine Primzahl, wenn Sie durch keine Primzahl $< n$ teilbar ist.

(a) Schreiben Sie eine rekursive Prozedur

```
first : (int → bool) → int → int
```

die zu p und m das kleinste $n \geq m$ mit $pn = true$ liefert.

(b) Sei ein *Tester für* $m \in \mathbb{N}$ eine Prozedur `int→bool`, die für $x \in \mathbb{N}$ testet, ob x durch keine Primzahl $\leq m$ teilbar ist (Ergebnis *false*, wenn durch mindestens eine Primzahl $\leq m$ teilbar, sonst *true*). Schreiben Sie mithilfe der Prozedur `first` eine Prozedur

```
next : int → (int → bool) → int
```

die zu einer Primzahl m und einem Tester für m die kleinste Primzahl $> m$ liefert.

(c) Schreiben Sie eine Prozedur

```
join : (int → bool) → int → (int → bool)
```

die zu einem Tester für $m \in \mathbb{N}$ und zur ersten Primzahl $n > m$ einen Tester für n liefert.

(d) Schreiben Sie mithilfe von `next` und `join` eine rekursive Prozedur

```
prime' : int → (int → bool) → int → int
```

die zu einer Primzahl m, einem Tester für m, und einem $n \geq 1$ die n-te Primzahl $\geq m$ liefert. (Beispiel: die 3. Primzahl ≥ 11 ist 17.)

(e) Schreiben Sie eine Prozedur

```
prime : int → int
```

die zu $n \geq 1$ die n-te Primzahl liefert.