Fachrichtung 6.2 — Informatik
Universität des Saarlandes
Tutorenteam der Vorlesung Programmierung 1

Programmierung 1 (Wintersemester 2012/13)

Aufgaben aus den Übungsgruppen 2

(Kapitel 2, Kaskadierung)

Hinweis: Dieses Übungsblatt enthält von den Tutoren für die Übungsgruppe erstellte Aufgaben.

Die Aufgaben und die damit abgedeckten Themenbereiche sind für die Klausur weder relevant, noch irrelevant.

1 Freie Bezeichner

Aufgabe 2.1
Schreiben Sie eine Prozedur, die einen freien Bezeichner enthält, aber auch in der leeren Umgebung ausgeführt werden könnte (wenn der Interpreter freie Bezeichner zuließe).

Aufgabe 2.2
Identifizieren Sie alle freien Bezeichner in folgendem Programm:

```
fun f (x:int) = a  + x

val a = 7
fun g (x:int) (y:int) :int = f(x) + f(y) + b
val b = c
val c = 7
```

Aufgabe 2.3 (*Zinkel, Zinkel, Zinkel...*)

Welche Umgebung liefert die Ausführung des folgenden Programmes in der leeren Umgebung?

```
val zinkel = ~37
fun g (x:int) (y:int) = x + y + zinkel
val zinkel = 0
fun zinkel' (a:int) = g zinkel
val x = zinkel' 7 3
```

2 Semantische Äquivalenz

Aufgabe 2.4
Warum nennen wir zwei Prozeduren semantisch äquivalent, wenn sie ich in Bezug auf statische und dynamische Semantik nach außen hin gleich verhalten? Warum nennen wir sie nicht äquivalent, wenn sie z.B. bei der Ausführung ihrer Deklaration die gleiche Umgebung liefern oder sie bis auf Umbenennung der Bezeichner gleich sind?

Aufgabe 2.5 (*Semantische Zulässigkeit*)

(a) Nennen Sie zwei Bedingungen, die erfüllt sein müssen, damit ein Programm semantisch zulässig ist.

(b) Finden Sie je ein Beispielprogramm das nur genau eine der Bedingungen nicht erfüllt.

(c) Schreiben Sie je zwei Programme, die nicht wohlgetypt sind. Tauschen Sie diese mit Ihrem Nachbarn aus und finden Sie dann in den neuen Programmen die nicht wohlgetypten Stellen.

(d) Diskutieren Sie: Ist es sinnvoll, bei semantisch unzulässigen Programmen von semantischer Äquivalenz zu reden?

Aufgabe 2.6
Geben Sie je zwei syntaktisch verschiedene, aber semantisch äquivalente Prozedurdeklarationen an, die die folgenden Funktionen berechnen:

Beispiel: $f\colon \mathbb{Z} \to \mathbb{Z}, f(x) = x$ wird berechnet durch die folgenden beiden Prozeduren

```
fun f (x:int) = x fun g (x:int) = 2 * x - x
```

Hinweis: Benennen Sie die Prozeduren nicht einfach um. Achten Sie darauf, tatsächlich verschiedene Prozedurrümpfe zu haben.

(a) $f\colon \mathbb{Z} \to \mathbb{Z}, f(x) = (x-1)^3$

(b) $f\colon \mathbb{R} \to \mathbb{R}, f(x) = x + 5.0$

(c) $f\colon \mathbb{Z} \to \mathbb{R}, f(x) = 4.0$

Aufgabe 2.7 (*Offizielle Übungsaufgabe aus dem Wintersemester 2011/12*)
Entscheiden Sie, ob folgende Programme jeweils semantisch quivalent sind:

a)
```
fun a (x:int) = x + a
```
```
fun a(x:int) = a + x
```

b)
```
fun fac (x:int) =
    if x = 0 then 1
    else fac(x-1)
fun b (p:int*int) = (#2p,#1p)
val it = b(fac(~1),fac(1))
```
```
fun fac (x:int) =
    if x = 0 then 1
    else x * fac(x-1)
fun b(p:int*int) = p
val it = b(fac(1),fac(~1))
```

c)
```
val it = 111111111111 mod 1
```
```
val it = 111111 mod 1
```

d)
```
fun di (x:int) = di (x-1)
```
```
fun di (x:int) = 1 + di (x-1)
```

3 Ausdrücke vs. Deklarationen

Aufgabe 2.8
Prozedurdeklarationen sind keine Ausdrücke. Deshalb ist z.B. für Aufgabe 2.7 des Übungsblattes

```
fun f (x:int) = x * x
```

keine gültige Lösung. Stattdessen ist

```
let fun f (x:int) = x * x in f end
```

eine gültige Lösung.

Ist dieses Vorgehen für alle Prozeduren möglich? Probieren Sie anhang einiger Beispiele aus, ob sie zu jeder Prozedur so einen Ausdruck erhalten, der semantisch zulässig ist und die gleiche Funktion berechnet.

Aufgabe 2.9
Aufgaben wie 3.7 des Übungsblattes sind typische Aufgabenstellungen. Warum wertet der Ausdruck zu einer Prozedur aus? Was müsste in der Prozeduranwendung noch übergeben werden, damit der Ausdruck zu einem Wert auswertet?
Überlegen Sie sich anhand dieser Aufgabe, wie man herausfindet, zu welchem Wert oder welcher Prozedur ein Ausdruck auswertet.

4 Kaskadierung

Aufgabe 2.10
Schreiben Sie die folgenden Prozeduren kaskadiert auf. Erläutern Sie Vorteile, die diese Darstellung bringt.

Beispiel:

```
fun f (g:int -> bool,x:int) :bool= g(x)
```

Für jeden neuen Wert von x, selbst wenn die Prozedur g gleichgelassen wird, muss $f(g, x)$ neu aufgerufen werden. Es ist viel praktischer, einmal g zu übergeben und dann für jeden neuen Wert von x nur noch mit diesem Aufrufen zu müssen. Dies ist möglich durch:

```
fun f (g:int->int) (x:int) = g(x)
fun g ...
val a = f g
val b = a 5
val c = a 7
val d = a 10345678
```

(a) fun mul (x:int,y:int) = x * y

(b) fun konditional (f: int * int -¿bool, x:int, y:int) = if f(x,y) then x else y

Aufgabe 2.11
Betrachten Sie die folgenden Deklarationen:

```
val even = fn x => x mod 2 = 0
fun odd n = not (even n)
fun greater m n = n > max
```

Dieter Schlau möchte eine Prozedur schreiben, die alle natürlichen Zahlen kleiner gleich n aufsummiert, für die ein gegebenes Prädikat p erfüllt ist. Weil er die Prozedur nicht für jedes mögliche Prädikat neu schreiben will, sucht er einen guten Ausweg. Wie kann er vorgehen?

Verwenden Sie die oben angegebenen Prädikate und ihre Summierungsprozedur, um eine Prozedur $sumEven : int \to int \to int$ zu schreiben, die alle natürlichen, geraden Zahlen in einem Bereich von a bis b summiert!

5 Knobelaufgaben

Aufgabe 2.12
Schreiben Sie zwei Prozeduren

$$cas : (int * int \to int) \to int \to int \to int$$
$$car : (int \to int \to int) \to int * int \to int$$

so dass *cas* zur kartesischen Darstellung einer zweistelligen Operation die kaskadierte Darstellung und *car* zur kaskadierten Darstellung die kartesische Darstellung liefert. Erproben Sie *cas* und *car* mit einem Interpreter!

Aufgabe 2.13
Betrachten Sie das folgende Programm:

```
val n = 2
fun f (x:int) = n*x
val n = 3
val g = fn (x:int) => (f x) * n
val n = 4
val h = f
```

(a) Wozu wertet $f\ 5$ aus?

(b) Wozu wertet $g\ 5$ aus?

(c) Woran bindet dieses Programm n, f, g und h?

Aufgabe 2.14
Wie sieht die Tripeldarstellung der Prozedur

```
fun f f = f 5
```

aus? Mit was wird die Prozedur aufgerufen?