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

Programmierung 1 (Wintersemester 2012/13)

Lösungsblatt 2

(Kapitel 2)

Hinweis: Dieses Zusatzübungsblatt wird von den Tutoren der Vorlesung "Programmierung 1" (WS 2012/13) erstellt. Es enthält unter anderem Aufgaben von den Übungsblättern der gleichen Vorlesung aus dem Wintersemester 2011/12 gehalten von Prof. Hermanns, sowie des Nachklausurtutoriums aus dem gleichen Jahr. Außerdem enthält es einige neue, von den Tutoren erstellte Aufgaben.

Die hier gestellten Aufgaben und die damit abgedeckten Themenbereiche sind weder für die Klausur relevant, noch irrelevant. Sie dienen vor allem dazu Ihnen andere Blickrichtungen auf den Stoff zu präsentieren.

Aufgabe 2.1 (*Offizielle Übung aus dem Wintersemester 11/12*)

Schreiben Sie eine Prozedur, die zwei natürliche Zahlen nach dem Schulalgorithmus multipliziert. Dazu ein Beispiel:

```
123 x 321
_____________________

   369
    246
     123
_____________________

   39483
```

Zuerst wird also $3 \cdot 123$ gerechnet, dann $2 \cdot 123$ und schließlich $1 \cdot 123$. Um dann das gewnschte Ergebnis zu erhalten, addiert man $3 \cdot 123 \cdot 100 + 2 \cdot 123 \cdot 10 + 1 \cdot 123$.

Lösung 2.1:

```
fun rev (n:int, a:int) :int = if n<10 then (a*10 + n) else rev ((n div 10), (a*10 + n mod 10))
fun mul'(n:int, m:int, a:int) :int = if m<=0 then a + n * m else mul' (n, (m div 10), (a*10 + n*(m
mod 10)))
fun mul (n:int, m:int) :int = if((m mod 10) = 0) orelse ((n mod 10) =0) then n * m else mul'(n,
(rev (m, 0)), 0)
```

Aufgabe 2.2 (*Offizielle Übung aus dem Wintersemester 11/12*)

Schreiben Sie eine Prozedur $divneg : int \to int$, die eine negative Zahl verdoppelt und bei postiven Zahlen als Argument divergiert. Bei Null als Argument soll 42 zurückgegeben werden.

Lösung 2.2:

```
fun divneg (n:int) :int = if (n < 0) then 2 * n else if (n = 0) then 42 else divneg(n)
```

Aufgabe 2.3 (*Offizielle Übung aus dem Wintersemester 11/12*)

Betrachten Sie erneut die Syntaxübersicht von SML auf Seite 33. Überlegen Sie sich nun eine Reihe von syntaktisch nicht zulässigen Ausdrücken und geben Sie die Gleichung an, gegen die Ihre Ausdrücke verstoßen.

Aufgabe 2.4 (*Für Fortgeschrittene, offizielle Übung aus dem Wintersemester 11/12*)

Geben Sie mit Hilfe der Syntaxübersicht von SML die ausführlichen syntaktischen Gleichungen für eine Teilsprache von SML an, die nur die vier arithmetischen Operationen $+, -, *$ und / und Konstanten kennt.

Lösung 2.4:

⟨Ausdruck⟩ ::= ⟨Konstante⟩
| ⟨Operatoranwendung⟩
| (⟨Ausdruck⟩)

⟨Operatoranwendung⟩ ::= ⟨Ausdruck⟩⟨Operator⟩⟨Ausdruck⟩

Aufgabe 2.5 (*Offizielle Übung aus dem Wintersemester 11/12*)

Betrachten Sie das folgende Programm:

```
val x = 3 + 2
fun f (y:int) = x + y
fun g (y:int):int = if y < x then 0 else y + g (y-1)
```

(a) Welche Umgebung liefert die Ausfhrung des Programms in der leeren Umgebung?

(b) Geben Sie die Umgebung an, in der der Rumpf der Prozedur *f* bei der Ausfhrung des Aufrufs *f 7* ausgefhrt wird.

(c) Geben Sie die Umgebung an, in der der Rumpf der Prozedur *g* bei der Ausfhrung des Aufrufs *g 13* ausgefhrt wird.

Lösung 2.5:

(a) [x := 5,
f := (fun f y = x+y, int → int, [x := 5]),
g := (fun g y = if y < x then 0 else y + g(y 1), int → int, [x := 5])]

(b) [x := 5, f := (fun f y = x + y, int → int, [x := 5]), y := 7]

(c) [x := 5, g := (fun g y = if y < x then 0 else y + g(y 1), int → int, [x := 5]), y := 13]

Aufgabe 2.6 (*Offizielle Übung aus dem Wintersemester 11/12*)

Geben Sie die freien Bezeichner der folgenden Programme an und zeigen sie durch Pfeile an, wo die Bezeichner gebunden werden. Welche der folgenden Programme sind geschlossen?

a)
```
val a = 5
val b = 10
fun f (x:int) = a + b + c
val c = 3
```

b)
```
val a = 0
val b = 0
val c = 3
fun f(x:int) = a + b + c
```

c)
```
val c = 3
val c = let
   val a = 4
   val b = 5
   fun f (x:int) = a * x * x + b * x + c
in
  f(4)
end
```

d)
```
fun f (x:int) =
  if x > 4
    then
      let
        val a = 4
        val b = 5
        val c = 3
      in
        f(4)
      end
    else
      a*x*x + b*x + c
```

Lösung 2.6:

a) Der Bezeichner c ist nicht gebunden, damit ist das Programm nicht geschlossen

b) Alle Bezeichner sind gebunden und das Programm ist geschlossen.

c) Alle Bezeichner sind gebunden und das Programm ist geschlossen.

d) Die Bezeichner a, b und c sind nicht gebunden, daher ist das Programm nicht geschlossen.

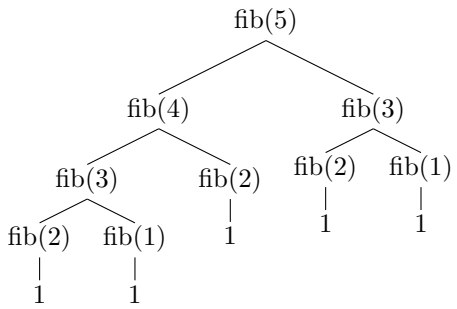
Aufgabe 2.7 (*Etwas kniffliger, aber sehr sinnvoll!, offizielle Übung aus dem Wintersemester 11/12*)

Sei die folgende Prozedur gegeben:

```
fun fib (n:int):int =
  if n=0
    then 0
    else if n <= 2
           then 1
           else fib(n-1) + fib(n-2)
```

Geben Sie ein verkürztes Ausführungsprotokoll für den Aufruf $fib(5)$ an. Überlegen Sie sich dabei auch, wie Sie die Rekursion geeignet darstellen.

Lösung 2.7:



Achtung: Es handelt sich bei der Lösung um eine Mischung aus Rekursionsfolge und verkürztem Ausführungsprotokoll.

Aufgabe 2.8 (*offizielle Übung aus dem Wintersemester 11/12*)

Geben Sie die Tripeldarstellung der folgenden Prozedur an.

```
fun fak (n:int):int = if n < 1
                      then 1
                      else n * fak (n-1)
```

Lösung 2.8:

```
(fun fak n = if n<1 then 1 else n * fak(n-1), int -> int, [])
```

Aufgabe 2.9 (*offizielle Übung aus dem Wintersemester 11/12*)

Entscheiden Sie von den folgenden vier Programmen, ob sie korrekt sind. Orientieren Sie sich dabei an den vier Ausführungsphasen eines Interpreters und geben Sie an, in welcher Phase die Ausführung gegebenfalls scheitert.

a)
```
val a = 5
fun f (x:int) = x + a
```

b)
```
val a = true
fun f (x:int) = x + a
```

c)
```
val a = 5
funn f (x:int) = x + a
```

d)
```
val a = 5
fun f (x:int) = a +
```

Lösung 2.9:

a) Dieses Programm ist korrekt und wird alle vier Phasen durchlaufen.

b) Dieses Programm wird in der *semantischen Analyse* scheitern, da die Addition nicht für ein Wertepaar aus *int* und *bool* definiert ist.

c) Diese Zeichenfolge scheitert in der *syntaktischen Analyse*, da wegen des $funn$ kein Syntaxbaum aufgebaut werden kann.

d) Diese Zeichenfolge scheitert in der *syntaktischen Analyse*. Es ist kein gültiges Programm, da der zweite Operand des +-Operators fehlt.

Aufgabe 2.10 (*Schriftliche Aufgabe aus dem Wintersemester 11/12*)
Hinweis: Die folgende Aufgabe soll einen anderen Blick auf die Programmierung anregen. Diskutieren Sie den Inhalt der Aufgabe am besten auch mit Ihren Kommilitonen.

Erklären Sie mit Hilfe eines oder auch mehrerer Vergleiche, was Rekursion ist. Beginnen Sie jeden Vergleich mit *"Rekursion ist wie . . . "*, und erklären Sie dann, weshalb Ihr Vergleich zutreffend ist. Seien Sie in der Wahl Ihrer Vergleiche kreativ, und lassen Sie sich nicht abhalten, Tätigkeiten und Gegenstände des alltäglichen Lebens zu verwenden.