14. 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, 10. Februar 2003

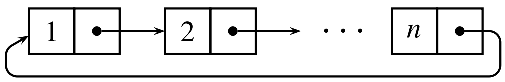
Schicken Sie Ihre Lösungen per E-Mail an Ihren Übungsgruppenleiter. Geben Sie in der E-Mail zuerst ihren Namen und die Nummer Ihrer Übungsgruppe an. Bitte reichen Sie alle Lösungen mit nur einer E-Mail ein.

Aufgabe 14.1: Imperative Listen $(25 = 5 + 10 + 10)$ Seien imperative Listen so wie im aktualisierten Kapitel 13 des Skripts definiert.

(a) Schreiben Sie eine Prozedur

```
circle : int → int ilist
```

die zu $n \geq 1$ eine zyklische Liste mit n Knoten liefert, die aufsteigend mit den Zahlen $1, \ldots, n$ markiert sind:



Die Prozedur soll den Zeiger auf den mit 1 markierten Knoten liefern.

(b) Schreiben Sie eine Prozedur

```
sizeR : 'a ilist → int
```

die die Anzahl der Referenzen einer imperativen Liste liefert. Die Prozedur soll auch für zyklische Listen funktionieren. Verwenden Sie eine Hilfsprozedur `sizeR'`, die sich in einem Akkumulatorargument alle besuchten Referenzen merkt.

(c) Schreiben Sie eine Prozedur

```
sizeN : 'a ilist → int
```

die die Anzahl der Knoten einer imperativen Liste liefert.

Aufgabe 14.2: Größte gemeinsame Teiler mit V ($20 = 10 + 10$) Die Prozedur

```
fun gcd(x,y) = if x<>y then
                  if x<=y then gcd(x, y-x) else gcd(x-y, y)
               else x
```

berechnet den größten gemeinsamen Teiler zweier positiven ganzen Zahlen.

(a) Schreiben Sie mithilfe einer Schleife eine nichtrekursive Variante `gcdi` der Prozedur `gcd`.

(b) Schreiben Sie eine Befehlssequenz *gcdc* ohne Prozedurbefehle, sodass das V-Programm

`[con` *x*`, con` *y*`] @` *gcdc* `@ [halt]`

den größten gemeinsamen Teiler zweier positiven ganzen Zahlen x und y berechnet.

Aufgabe 14.3: Übersetzung und Rückübersetzung ($15 = 5 + 10$) Seien die folgenden Ausdrücke gegeben:

```
datatype exp =
    Con    of int             (* constant          *)
  | Add    of exp * exp       (* addition          *)
  | Sub    of exp * exp       (* subtraction       *)
  | Mul    of exp * exp       (* multiplication    *)
```

(a) Schreiben Sie eine Prozedur

compile : exp → code

die einen Ausdruck in ein V0-Programm übersetzt, dessen Ausführung den Wert des Ausdrucks liefert.

(b) Schreiben Sie eine Prozedur

decompile : code → exp

sodass für alle Ausdrücke e gilt:

`decompile(compile` *e*`) =` *e*

Aufgabe 14.4: Endaufrufe (5) Markieren Sie in den folgenden Prozedurdeklarationen alle Endaufrufe.

```
fun f(x,y) = if x<y then y else f(x+1,y)

fun g x = if x<0 then x else f(x, 2*x)

fun h (1::xs) = h xs
  | h xs      = p xs

and p (2::xs) = h xs
  | p (x::xs) = g x + 5
```

Aufgabe 14.5: Fibonacci-Prozedur in V ($20 = 10 + 10$)

(a) Übersetzen Sie die Prozedur

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

in eine Befehlssequenz für V. Nehmen Sie dabei an, dass die Befehlssequenz mit der Adresse 0 beginnend im Programmspeicher abgelegt wird.

(b) Übersetzen Sie die Prozeduren

```
fun fibi'(n,a,b) = if n<=0 then a else fibi'(n-1, b, a+b)

fun fibi n = fibi'(n,0,1)
```

in eine Befehlssequenz für V. Nehmen Sie dabei an, dass die Befehlssequenz mit der Adresse 0 beginnend im Programmspeicher abgelegt wird. Übersetzen Sie alle Endaufrufe mit `callR`.

Aufgabe 14.6: Verschränkte Rekursion in V (15) Übersetzen Sie die Prozeduren

```
fun even x = if x=0 then true else odd(x-1)
and odd  x = if x=0 then false else even(x-1)
```

in eine Befehlssequenz für V. Nehmen Sie dabei an, dass die Befehlssequenz mit der Adresse 0 beginnend im Programmspeicher abgelegt wird. Übersetzen Sie alle Endaufrufe mit `callR`.

Aufgabe 14.7: Syntax und Semantik von W (freiwillig, keine Abgabe) Definieren Sie die abstrakte Syntax von W mithilfe einer abstrakten Grammatik. Definieren Sie die statische und die dynamische Semantik von W mithilfe von Inferenzregeln. Definieren Sie eine konkrete Syntax für W. Schreiben Sie einen Lexer und einen Parser für W.