15. Übungsblatt zu Programmierung 1, WS 2008/09

Prof. Dr. Gert Smolka, Mark Kaminski, M.Sc.
www.ps.uni-sb.de/courses/prog-ws08/

Lesen Sie im Buch: Kapitel 16

Aufgabe 16.3 (Übersetzung) Schreiben Sie eine Prozedur *compile* : *exp* → *code*, die einen Ausdruck in ein Maschinenprogramm übersetzt, das den Ausdruck auswertet. Legen Sie dabei die folgenden Ausdrücke zugrunde:

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

Aufgabe 16.4 (Arithmetische Maschine) Schreiben Sie eine endrekursive Prozedur *run* : *int list* → *code* → *int*, sodass *run nil* (*compile e*) den Wert des Ausdrucks *e* liefert. Das erste Argument von *run* soll den für die Ausführung des Maschinenprogramms benötigten Stapel darstellen (oberstes Element des Stapels als erstes Element der Liste). Für Maschinenprogramme, die nicht mit *compile* darstellbar sind, soll *run* die Ausnahme *Domain* werfen.

Aufgabe 16.5 (Rückübersetzung) Wenn man arithmetische Ausdrücke nach dem beschriebenen Schema in Maschinenprogramme übersetzt, geht keine Information verloren. Also ist eine Rückübersetzung der Maschinenprogramme in die Ausdrücke möglich. Dies gelingt mit einer Maschine, die auf ihrem Stapel die bereits erkannten Teilausdrücke ablegt. Schreiben Sie eine entsprechende Prozedur *decompile* : *code* → *exp*. Für Maschinenprogramme, die nicht mit *compile* darstellbar sind, soll *decompile* die Ausnahme *Domain* werfen.

Hilfestellung: Schreiben Sie *decompile* mithilfe einer an der Prozedur *run* aus Aufgabe 16.4 angelehnten Prozedur *decompile′* : *exp list* → *code* → *exp*.

Aufgabe 16.6 (Baumrekonstruktion aus der Postlinearisierung) Die Übersetzung von arithmetischen Ausdrücken in Code für M entspricht bis auf die Reversion der Argumente der Postlinearisierung von Bäumen (§ 7.6.3). Dementsprechend können wir einen Baum aus seiner Postlinearisierung rekonstruieren, indem wir die Postlinearisierung mit einer Maschine ausführen, die auf ihrem Stapel die bereits erkannten Teilbäume ablegt.

a) Schreiben Sie eine Prozedur *post* : *tree* → *int list*, die die Postlinearisierung eines Baums liefert.

b) Schreiben Sie eine Prozedur *depost* : *int list* → *tree*, die mit *post* übersetzte Bäume rückübersetzt. Verwenden Sie dabei eine endrekursive Hilfsprozedur *depost′* : *tree list* → *tree list* → *int list* → *tree*, die die bereits gebildeten Bäume und eine Unterbaumliste als Akkus erhält.

c) Schreiben Sie eine Prozedur *depre* : *int list* → *tree*, die die Prälinearisierung von Bäumen rückübersetzt. Reversieren Sie dazu die Prälinearisierung und verwenden Sie eine Variante der Prozedur *depost′*, die beim Bilden eines Baums die Unterbaumlisten reversiert.

d) Vergleichen Sie die Rekonstruktion von Bäumen in dieser Aufgabe mit der auf rekursivem Abstieg beruhenden Rekonstruktion in Aufgabe 13.7 auf S. 266. Machen Sie sich klar, dass das hier verwendete stapelbasierte Verfahren im Gegensatz zum rekursiven Abstieg nur Endrekursion benötigt.

Aufgabe 16.9 Geben Sie ein möglichst kurzes Programm p an, sodass das Programm *con k* :: p für positives k terminiert und für nicht positives k divergiert.

Aufgabe 16.10 Schreiben Sie ein Programm, das zu x und n mit einer Schleife die Potenz x^n berechnet. Schreiben Sie das Programm so wie oben gezeigt erst in W und dann in M. Achten Sie darauf, dass Ihr Programm nur die Potenz x^n im Stapel zurücklässt.

Aufgabe 16.12 Nehmen Sie an, dass der Programmspeicher der Maschine wie folgt belegt ist:

0	1	2	3	4	5	6	7	8	9	10
con 2	*getS 0*	*con 0*	*leq*	*cbranch 6*	*con 1*	*getS 0*	*sub*	*putS 0*	*branch ~8*	*halt*

Simulieren Sie die Ausführung des Programms mit Papier und Bleistift.

a) Geben Sie die Folge der Adressen an, die der Programmzähler durchläuft. Hilfe: Die Folge beginnt mit 0 und endet mit 10. Sie hat insgesamt 33 Elemente.

b) Wird jeder Befehl des Programms mindestens einmal ausgeführt?

c) Geben Sie die Adressen im Programmspeicher an, die genau einmal zur Ausführung kommen.

d) Skizzieren Sie das Maschinenprogramm durch ein Programm in W.

Aufgabe 16.16 Schreiben Sie ein Maschinenprogramm, das die Länge einer Liste berechnet. Halten Sie sich an die folgende Skizze in W:

```
var xs
var n := 0
while 0<=xs do
   n := n+1 ;
   xs := xs.1
end
return n
```

Der Ausdruck *xs.1* liefert den Wert in der ersten Zelle des Blocks, dessen Adresse der Wert der Variablen *xs* ist.

Aufgabe 16.20 Erweitern Sie den Übersetzer für W1 um Eingabevariablen. Erweitern Sie dazu die in Abbildung 16.3 auf S. 337 gezeigte Programmdarstellung um eine Komponente für die Liste der Eingabevariablen. Sorgen Sie dafür, dass die Ausgabeanweisung auch die Zellen für die Eingabevariablen deralloziert.

Aufgabe 16.24 Übersetzen Sie die folgende Prozedur nach M:

```
fun fac n = if n<2 then 1 else fac(n-1)*n
```

Nehmen Sie dabei an, dass die Befehlssequenz für die Prozedur im Programmspeicher mit der Adresse 0 beginnt.

Aufgabe 16.25 Übersetzen Sie die folgende Prozedur nach M:

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

Aufgabe 16.26 Übersetzen Sie die folgende Prozedur nach M:

```
fun revi (nil, ys) = ys
  | revi (x::xr, ys) = revi (xr, x::ys)
```

Aufgabe 16.31 Markieren Sie in den folgenden Prozedurdeklarationen alle Prozeduranwendungen in Endposition:

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

fun g x = if x<0 then x else 1+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 16.32 Stellen Sie den Zustand der Maschine M dar, der bei der Ausführung des Programms *gcdc*@[*con 16*, *con 12*, *call 0*, *halt*] vorliegt, nachdem der Endaufrufbefehl zum zweiten Mal ausgeführt wurde. Dabei bezeichnet *gcdc* den oben gezeigten Code für die Prozedur *gcd*. Orientieren Sie sich an der Darstellung in Abbildung 16.5 auf S. 343.

Aufgabe 16.33 Formulieren Sie die Prozeduren

```
fun fac'(a,n) = if n<=1 then a else fac'(a*n,n-1)
fun fac n = fac'(1,n)
```

in M. Achten Sie darauf, dass Sie nur Endaufrufbefehle verwenden.