Max Wisniewski , Alexander Steen

Tutor: Ansgar Schneider

Aufgabe 1

Definieren Sie die WSKEA-Maschine derart um, dass bei arithmetischen Ausdrücken rechte Unterausdrücke vor linken ausgewertet werden. Konstruieren Sie ein Beispiel, für das ein abweichendes Ergebnis erzielt wird.

Lösung:
Dafür müssen lediglich zwei Zustandübergänge geändert werden, der Rest kann gleich bleiben:
(i) Zuerst müssen die rechten Ausdrücke einer Operation zuerst auf den Keller gelegt werden, damit diese zuerst ausgewertet werden:

$\Delta < W|S|T_1\underline{OP}T_2.K|E|A >:=< W|S|T_2.T_1.\underline{OP}.K|E|A >$

(ii) Dann müssen wir beim Herunternehmen der Ergebnisse die korrekte der Operationsanwendung wieder herstellen:

$\Delta < n_1.n_2.W|S| + .K|E|A >:=< n_1 + n_2.W|S|K|E|A >$, falls $n_1 + n_2$ darstellbar.

Diese Regel kann analog auf alle anderen arithmetische Operationen angewendet werden.
Abweichendes Beispiel:

Links-vor-Rechts:

$$\begin{aligned}\Delta < ()|()|\underline{read} - \underline{read}|(3,2)|A > &:= \Delta < ()|()|\underline{read}, \underline{read}, -|(3,2)|A > \\ &:= \Delta < (3)|()|\underline{read}, -|(2)|A > \\ &:= \Delta < (2,3)|()| - |()|A > \\ &:= \Delta < (3-2)|()| - |()|A >\end{aligned}$$

Rechts-vor-Links:

$$\begin{aligned}\Delta < ()|()|\underline{read} - \underline{read}|(3,2)|A > &:= \Delta < ()|()|\underline{read}, \underline{read}, -|(3,2)|A > \\ &:= \Delta < (3)|()|\underline{read}, -|(2)|A > \\ &:= \Delta < (2,3)|()| - |()|A > \\ &:= \Delta < (2-3)|()| - |()|A >\end{aligned}$$

Im ersten Fall wird im linken Ausdruck die 3 zuerst gelesen und im rechten die 2. Wir rechnen vollglich $3-2$ und erhalten eine 1. Im zweitenn Fall gehen wir erst in den rechten Ausdruck, lesen dort die 3 und gehen danach in den linken zweig und lesen dort als nächstes die 2. Wir rechnen also $2-3$ und erhalten eine -1.

Aufgabe 2

Erweitern Sie die WSKEA-Maschine um eine Komponente `N` für Nachrichten (Texte), in der kurze sinnvolle Meldungen eingetragen werden, wenn es keinen Folgezustand gibt oder wenn die Ausführung korrekt terminiert.

Lösung:

Wir erweitern Zunächst unseren Zustand z um eine Nachricht. Da wir nur Nachrichten zurückgeben, wenn wir abstürzen oder fertig sind, brauchen wir keinen Stack und keine Queue. Es muss einfach ein Text gespeichert werden. $< W|S|K|E|A|N >$, dabei sind W, S, L, E, A wie in der ursprünglichen WSKEA Maschine und N ist das eben erwähnt Feld für die einzelne Nachricht. Der Startzustand $z_{P,E} =< \varepsilon|S_0|P.\varepsilon|E|\varepsilon|\varepsilon >$ ist äquivalent zum ursprünglichen Startzustand. Der letzte Eintrag für die Nachricht N ist einfach leer. Da wir nun auch im Fehlerfall eine Überführung haben wollen und nicht einfach abbrechen, definieren wir eine Funktion, die die unendliche Ausführung ermöglicht und bei Beendigung Idempotent sein sollte:

$$\Delta < W|S|K|E|A|n >=< W|S|K|E|A|n >, \quad \text{für } n \neq \varepsilon$$

Dies bedeutet, sollten wir eine Nachricht empfangen haben, lassen wir den Zustand unverändert, da es sich um einen Fehler handelt oder das Programm schon fertig ist. Damit wir nicht so viel schreiben müssen übernehmen wir alle anderen Überführungsfunktionen, wie gehabt, wenn die nachricht ε ist. Nun müssen wir nur noch Fälle betrachten in denen das Programm beendet wird oder ein Fehler auftritt.

$$\Delta < W|S|\varepsilon|E|A|\varepsilon >=< W|S|\varepsilon|E|A| \text{ Programm erfolgreich beendet } >$$

Einen Fehler können vielfällitg auftreten, wie wir auf dem ersten Übungsblatt gesehen haben, wir werden es hier exemplarisch für IF und $+$ zeigen

$$\Delta < n1.n2.W|S|+.K|E|A|\varepsilon >=< W|S|K|E|A|\text{falsche Typ bei } + >, \quad n1, n2 \text{ keine Zahl}$$

$$\Delta < n1.\varepsilon|S| + .K|E|A|\varepsilon >=< W|S|K|E|A|\text{Zu wenig Zahlen für } + >$$

$$\Delta < b.W|S|if.p1.p2.K|E|A|\varepsilon >=< W|S|K|E|A|\text{Bedingung ist keine Zahl} >, \quad b \text{ kein Bool}$$

Aufgabe 3

Die Syntax von WHILE sei um das repeat-until-Konstrukt erweitert, wie in der Aufgabe. Ergänzen Sie die operationelle Semantik.

Lösung:

Die übliche Semantik von REPEAT C UNTIL B ist, dass man C solange ausführt, wie B gilt. Der unterschied zu While ist, dass man C zunächst einmal ausführt, bevor man die Bedingung prüft. Um es un leicht zu machen, formen wir REPEAT deshalb auf WHILE um. Die Berechnung wird dadurch nicht verändert.

$$\Delta < W|S|\underline{REPEAT}\, C\, \underline{UNTIL}\, B.K|E|A >=< W|S|C.\underline{WHILE}\, \underline{NOT}\, (B)\, \underline{DO}\, C\, \underline{OD}.K|E|A >$$

Nach dieser Umformung wird zunächst einmal C ausgeführt. Nach dieser Ausführung wird B geprüft. Ist es wahr, wird die Schleife abgebrochen. Wenn nicht geht es in einen neuen Durchlauf über.

Aufgabe 4

Implementieren Sie WSKEA in einer Sprache Ihrer Wahl.

Lösung:

Unsere Wahl war Haskell, da wir hier schon vorher die Datentypen implementiert haben.

Zunächst haben wir wieder die Datentypen für T,B,C,P und die Grundtypen importiert. Danach haben wir einen neuen Datentyp gebraucht, da Haskell stark getypt ist, damit wir sowohl die Kontrollstrukturen, als auch die While-Struckturen auf den Programmstack legen wollten.

```
data Con = WhileC | IfC | NotC | AssignC I | BOutC | TOutC
   deriving Show

data PK = WhileProgramm C | ControlOp OP | ControlBop BOP |
   ControlC Con | ControlB B | ControlT T deriving Show

type WSKEA = ([K],Map I Z, [PK], [K], [K])
```

Mit diesen Datentypen konnten wir WSKEA analog zur Definition in der Vorlesung als 5-Tupel anlegen.
Die Funktion anfang, nimmt ein While Programm und eine Liste von Eingaben und gibt des Starttupel zurück. Dabei ist der Wertekeller zu beginn leer und im Wörterbuch steht kein Eintrag. Dies Ausgabe hat auch noch keinen Wert ausgegeben.

```
anfang :: P ->  [K] -> WSKEA
anfang program eingabe = ([], Map.empty , [(WhileProgramm
   program)], eingabe, [])
```

Die Überführungsfunktion Δ ist nicht schwer zu Implementieren nur aufwendig. Da Haskell funktional ist und die Überführungsfunktion dies auch ist, kann man prinzipiell die Definition aus der Vorlesung und dem Buch abschreiben und auf Haskellsyntax umarbeiten. (Die Funktion wird einmal am Ende angefügt.)
Bevor die Funktion einmal zur gänze herunter geleiert wird einmal ein Test am divprog vom letzten mal einfach in ghci eingegeben:

```
let wska0 = anfang divTest [(LiteralInteger 10), (
   LiteralInteger 2)]
> ([],fromList [],[WhileProgramm (Seq (Assign "x" 'read') (Seq
   (Assign "y" 'read') (Seq (Assign "g" 0) (Seq (While (x>=y) (
   Seq (Assign "g" (g+1)) (Assign "x" (x-y)))) (TOut g)))))
   ],[10,2],[])

(iterate delta wska0) !! 100
> ([],fromList [("g",5),("x",2),("y",2)],[ControlT x,ControlT y
   ,ControlOp -,ControlC (AssignC "x"),ControlB (x>=y),ControlC
    WhileC,ControlB (x>=y),WhileProgramm (Seq (Assign "g" (g+1)
   ) (Assign "x" (x-y))),WhileProgramm (TOut g)],[],[])

(iterate delta wska0) !! 500
> ([],fromList [("g",5),("x",0),("y",2)],[],[],[5])
```

Die Erste Abgabe zeigt den Startwert der WSKA Maschine, bei Eingabe von 'divTest', der zweite zeigt den Zustand der Maschine nach 100 iterationen von delta. An dieser Stelle sind wir noch in der Berechnung, haben von gearbeitet und in den Variablen stehen schon Werte drin, die nicht vom laden kommen. Nach 500 Schritten sind wir fertig, dies wird schon vorher eingetreten sein, aber der Zustand ändert sich nicht mehr.

Delta:

```
delta :: WSKEA -> WSKEA
delta (w,s,[],e,a) = (w,s,[],e,a)
-- ControlC
delta (((LiteralBool True):w), s, ((ControlC WhileC):b:p:k),e,
   a)     = (w, s, (p:b:(ControlC WhileC):b:p:k), e, a)
delta (((LiteralBool False):w), s, ((ControlC WhileC):_:_:k), e
   , a)   = (w, s, k, e, a)
delta (((LiteralBool True):w), s, ((ControlC IfC):p1:_:k), e, a
   )    = (w, s, (p1:k), e, a)
delta (((LiteralBool False):w), s, ((ControlC IfC):_:p2:k), e,
   a)   = (w, s, (p2:k), e ,a)
delta (((LiteralBool b):w), s, ((ControlC NotC):k), e, a)
     = (((LiteralBool (not b)):w), s, k, e, a)
delta (((LiteralInteger v):w), s, ((ControlC (AssignC id)):k),
   e, a)  = (w, Map.insert id v s, k, e, a)
delta (((LiteralInteger v):w), s, ((ControlC TOutC):k), e, a)
       = (w, s, k, e, ((LiteralInteger v):a))
delta (((LiteralBool b):w), s, ((ControlC BOutC):k), e, a)
       = (w, s, k, e, ((LiteralBool b):a))
-- ControlBop und Op
delta (((LiteralInteger i1):(LiteralInteger i2):w), s, ((
   ControlBop bop):k), e, a)
  = (((LiteralBool (decodeBOP bop i2 i1)):w), s, k, e, a)
delta (((LiteralInteger i1):(LiteralInteger i2):w), s, ((
   ControlOp op):k), e, a)
  = (((LiteralInteger (decodeOP op i2 i1)):w), s, k, e, a)
-- ControlB Boolauswertung
delta (w,s,((ControlB (Literal b)):k),e,a)
  = (((LiteralBool b):w),s,k,e,a)
delta (w,s,((ControlB (Not b)):k), e, a)
  = (w, s, ((ControlB b):(ControlC NotC):k), e, a)
delta (w,s,((ControlB (BApp t1 bop t2)):k),e,a)
  = (w, s, ((ControlT t1):(ControlT t2):(ControlBop bop):k), e,
      a)
delta (w, s, ((ControlB BRead):k), ((LiteralBool b):e), a)
  = (((LiteralBool b):e), s, k, e, a)
-- ControlT Termauswertung
delta (w, s, ((ControlT (Z z)):k), e, a)
  = (((LiteralInteger z):w), s, k ,e ,a)
delta (w, s, ((ControlT (Id i)):k), e, a) =
  let v = Map.lookup i s in
    case v of
      (Just value)  -> (((LiteralInteger value):w), s, k, e, a)
      _       -> error "No transition defined"
delta (w, s, ((ControlT (TApp t1 op t2)):k), e, a)
  = (w, s, ((ControlT t1):(ControlT t2):(ControlOp op):k), e, a
     )
delta (w, s, ((ControlT (TRead)):k), ((LiteralInteger i):e), a)
  = (((LiteralInteger i):w), s , k, e, a)
-- WhileProgramm
delta (w, s, ((WhileProgramm Skip):k), e, a)
  = (w, s, k, e, a)
delta (w, s, ((WhileProgramm (Assign i t)):k), e, a)
  = (w, s, ((ControlT t):(ControlC (AssignC i)):k), e, a)
delta (w, s, ((WhileProgramm (Seq c1 c2)):k), e, a)
  = (w, s, ((WhileProgramm c1):(WhileProgramm c2):k), e, a)
delta (w, s, ((WhileProgramm (While b c)):k), e, a)
  = (w, s, ((ControlB b):(ControlC WhileC):(ControlB b):(
     WhileProgramm c):k), e, a)
delta (w, s, ((WhileProgramm (If b c1 c2)):k), e, a)
  = (w, s, ((ControlB b):(ControlC IfC):(WhileProgramm c1):(
     WhileProgramm c2):k), e, a)
delta (w, s, ((WhileProgramm (BOut b)):k), e, a)
  = (w, s, ((ControlB b):(ControlC BOutC):k), e, a)
delta (w, s, ((WhileProgramm (TOut t)):k), e, a)
  = (w, s, ((ControlT t):(ControlC TOutC):k), e, a)
-- undefined stuff
delta _ = error "No transition defined"
```