Übungsblatt 5

Abgabe: 4.06.2007

Vom Rechnen mit Bäumen

Wir erweitern die polnische Notation (vgl. Serie 3) zu einer Programmiersprache über den ganzen Zahlen (die übrigens Turing-äquivalent ist). Dazu wird die bekannte Grammatik wie folgt ergänzt/modifiziert:

```
binop  = '+' | '/' | '*'
minop  = '-'
lb     = '('
rb     = ')'
komma  = ','
nums   = num {num}
num    = '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'
assign = '='
compop = '==' | '!=' | '<' | '>' | '<=' | '>='
var    = 'a..z' {('a..z' | 'A..Z' | '0..9')}
label  = 'A..Z'  {('a..z' | 'A..Z' | '0..9')}
stop   = 'STOP'
ifop   = 'if'
seqop  = ';'
mark   = ':'
```

Ein Programm besteht aus einem Initialisierungsblock und aus eventuellen weiteren Blöcken. Im Initialisierungsblock dürfen nur konkrete Werte zugewiesen werden, in allen anderen Blöcken beliebige Ausdrücke. Jeder Block endet entweder mit einem `STOP` oder einer Verzweigung, die abhängig von der Auswertung einer Bedingung zu einem anderen Block springt. Damit gibt es leider mehr als nur fünf Regeln....

```
TERM    = BINTERM
        | MINTERM
        | nums
        | var

BINTERM = binop ARGS

MINTERM = minop MINREST

MINREST = ARGS
        | nums
        | var

ARGS    = lb TERM komma TERM rb

SIMPASS = assign lb var komma (nums | minop nums)  rb seqop
```

```
FULLASS = assign lb var komma TERM rb seqop

BOOL    =  compop  lb (var | nums | minop nums ) komma (var | nums | minop nums ) rb
COND    =  ifop lb BOOL komma label komma label rb

INITBLOCK = SIMPASS { SIMPASS } (stop | COND)
BLOCK     = label mark  FULLASS { FULLASS } (stop | COND)

PROGRAM  = INITBLOCK { BLOCK }
```

Dabei darf natürlich `'if'` nicht als Variable und `'STOP'` nicht als Label verwendet werden. Variablen dürfen im `INITBLOCK` nicht mehrfach verwendet werden.

Wir verarbeiten diese Programmiersprache in den folgenden drei Aufgabenteilen, die aufeinander aufbauen

Aufgabe 1 Lesen und Prüfen

Schreibt einen Syntaxchecker zur obigen Grammatik, der die erkannten Token zur Weiterverarbeitung auch speichert und zusätzlich die Anzahl der deklarierten Variablen im `INITBLOCK` sowie die Anzahl der weiteren Blöcke ermittelt. Es ist eine gute Idee, bereits an dieser Stelle das unäre Minus aufzulösen, und so auch negative Zahlen als `nums` zu behandeln.

Aufgabe 2 Erstellen des Kontrollflussgraphen

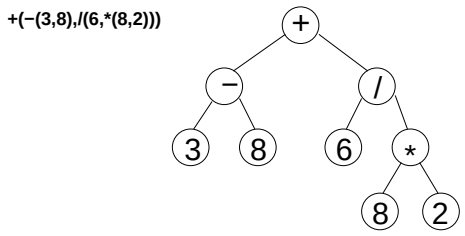
Aus den jeweils abschliessenden Bedingungen der Blöcke ergibt sich ein Kontrollflussgraph, der angibt, wie das Programm durchlaufen werden kann. Erstellt ein Programm zur Berechnung dieses Kontrollflussgraphen G, in dem die Knoten gerade die einzelnen Zuweisungsblöcke repräsentieren, in dem der `INITBLOCK` den Startknoten darstellt, und eine Transition von einem Knoten A zu einem Knoten B mit dem Label `bool` existiert, falls A mit `if(bool, B, L2)` oder `if(!bool,L1,B)` endet. Diese möglichen Transitionen können auch innerhalb eines Knotens implizit codiert werden.

Aufgabe 3 Ausführen von Baumprogrammen

Nachdem Aufgabe 2 beschreibt, wie sich das Programm von Block zu Block bewegen kann, soll jetzt das Programm ablaufen. Für jede Variable wird ein Baum angelegt, der den aktuelle Berechnung der Variablen auf der linken Seite in einem Assignment-Statement beschreibt Dabei wird die rechte Seite einer Zuweisung als Binärbaum dargestellt.

Falls innerhalb eines Baumes eine Variable vorkommt, so wird diese Variable durch ihren jeweiligen bisherigen aktuellen Baum ersetzt.

Am Ende eines Zuweisungsblockes werden die Bedingungen des Kontrollflussgraphen aus Aufgabe 2 ausgewertet, um den Folgeblock zu ermitteln. Bei einer `STOP`



- Anweisung werden alle Variablen ausgewertet und das Programm beendet. Dieser Interpreter besteht also aus einer Manipulations- und Evaluationskomponente für Berechnungsbäume und einer Kontrollkomponente, die den Lauf durch das Programm überwacht.