Theorie und Implementation objektorientierter Programmiersprachen
Theoretische Informatik, LMU München
Andreas Abel

Übungsblatt 2

Abgabe per E-mail an `abel@informatik.uni-muenchen.de`.
Abgabe bis: Montag, **6. Mai 2002, 10.15 Uhr**

Hinweis: Mit diesem Übungsblatt beginnt die Implementation des bislang theoretisch behandelten Stoffes. Sie dürfen für die Implementation eine beliebige strukturierte Programmiersprache benutzen. Empfohlen wird jedoch eine moderne, statisch getypte Sprache, die rekursive Datenstrukturen (algebraische Datentypen) und *pattern matching* (Definition durch Fallunterscheidung) unterstützt und den Hauptspeicher automatisch verwaltet (d.h. mit *garbage collection*). Beispiele für eine solche Sprache sind ML (SML, OCaml) und Haskell. Denkbar ist auch eine objektorientierte Sprache wie z.B. JAVA, in der algebraische Datenstrukturen durch Klassenhierarchien simuliert werden können.

Zur Abgabe der Lösungen: Die abgegebenen Programme müssen ausführbar sein und **bei Start alle geforderten Testfälle behandeln** und die Ergebnisse ausgeben.

Aufgabe 1: [Interne Syntax]

Finden Sie eine geeignete Repräsentation für die interne Syntax von Termen t (Skript Teil 2.2). Implementieren Sie eine Methode, die Terme in externe Syntax konvertiert (Druckfunktion), so dass sie ausgegeben werden können. Lassen Sie sich folgende Terme ausdrucken:

```
(len "vier") plus (3 minus 2)
str (4 minus (3 minus 1))
(str (len 4)) plus (len (str 4))
(4 minus (3 minus 1)) plus (len (str -4444))
```

Aufgabe 2: [Auswertung]

Implementieren Sie Auswertung von Ausdrücken durch die “big-step” Semantik. Testen Sie ihren Algorithmus an den Beispielen von Aufgabe 1.

(Bitte wenden!)

Aufgabe 3: [Partielle Operationen]

Wir erweitern unsere Sprache um Multiplikation, Division und Konvertierung von Strings in Zahlen. Externe Syntax:

$$
\begin{array}{rcl}
e & ::= & \ldots \\
& | & e_1 \texttt{ * } e_2 \\
& | & e_1 \texttt{ / } e_2 \\
& | & \texttt{Integer.parseInt}(e)
\end{array}
$$

Interne Syntax:

$$
\begin{array}{rcl}
t & ::= & \ldots \\
& | & e_1 \texttt{ times } e_2 \\
& | & e_1 \texttt{ div } e_2 \\
& | & \texttt{int } e
\end{array}
$$

Erweitern Sie die big-step Auswertungssemantik um Regeln für die neuen Termkonstruktoren. Achtung: Division und Konvertierung eines Strings in eine Zahl sind partielle Operationen!
[Auf Papier oder als elektronisches Dokument abzugeben.]

Aufgabe 4: [Erweiterung des Interpreters]

Erweitern Sie interne Syntax, Druckfunktion und Auswertungsalgorithmus, so dass ihr Interpreter die in Aufgabe 3 beschriebene Erweiterung der Sprache behandeln kann. Testen Sie den Interpreter mit den folgenden Ausdrücken:

```
(int (str (10 times 10))) div (-10)
(3 plus 4) div (3 minus 1)
3 times (int "bla")
5 div (1 minus (len (str (len "eins"))))
```

Viel Erfolg!