Lehrstuhl für Programmiersprachen und Übersetzerkonstruktion
CAU Kiel, Informatik und Prakt. Mathematik
Prof. Dr. M. Hanus, Dr. Frank Huch

24118 Kiel
Christan-Albrechts-Platz 4
Tel.: 0431 / 880-7277

6. Übung „Prinzipien von Programmiersprachen“

Bearbeitung bis zum 7. November 2004

Programm für die Übung am 2. Dezember

- Besprechung letzte Übung
- Module und Klassen

Aufgabe 1

Sei im Folgenden eine Methodendeklaration in der Umgebung von der Form:

$$m : method(x_1 : \tau_1, \ldots, x_n : \tau_n, \tau, S)$$

Hierbei sind $x_i : \tau_i$ die Parameter, τ der Typ des Rückgabewertes und S der Rumpf der Methode. Die Deklarationsumgebung wird für die Methodendeklaration nicht benötigt, da sie bereits in der Klassendeklaration abgelegt wurde.
Geben Sie eine Ableitung von $\langle E, M\rangle$ `y = x.m();` $\langle E, M'\rangle$ an, wobei

i) $E = E';\ x : (ref2, C);\ y : (ref3, \texttt{double})$

ii) $E' = C : class\{\texttt{double}\ x,\ \texttt{double}\ y,\ m : method(\emptyset, \texttt{double}, S), \emptyset\}$;

iii) $S = m$ `= x*x+y*y;`

iv) $M = \{ref2 \mapsto ref4,\ ref3 \mapsto 0.0,\ ref4 \mapsto 3.0,\ ref5 \mapsto 5.5\}$.

Wie sieht M' nach der Abarbeitung von `y = x.m();` aus? Die Herleitungen für $\overset{lookup}{\vdash}$ und $\overset{L}{\vdash}$ können weggelassen werden.

Aufgabe 2 **2 Punkte**

In dieser Aufgabe wollen wir den abstrakten Datentyp Stack in Java realisieren.

a) Definieren Sie einen abstrakten Datentyp Stack, welcher beliebige Objekte aufnehmen kann, und implementieren Sie seine Schnittstelle in Java (Klassendefinition mit leeren Methoden Rümpfen oder, wer es schon kennt, abstrakte Klasse).

b) Um zunächst eine einfache prototypische Implementierung zu haben, realisieren Sie den Stack als Array. Da sich die Größe des Stacks bei jeder *push*/*pop*-Operation ändert, müssen Sie jeweils ein neues Array von entsprechend veränderter Größe anlegen und das alte Array kopieren.

c) Verwenden Sie Ihre Klasse Stack zur Definition einer Klasse `Turingmaschine` für deterministische Turingmaschinen. Die Idee hierbei ist die geschickte Kombination zweier Stacks zur Realisierung des Bandes der Turingmaschine. Beachten Sie hierbei, daß am Ende des Bandes jederzeit Blanks angefügt werden können.

   Überlegen Sie sich geeignete Datenstrukturen zur Repräsentation von Zuständen, Startzustand, Endzuständen und Zustandsübergängen der Turingmaschine. Als Alphabet soll Ihre Turingmaschine Zeichen (`char`) verwenden. So können Sie die Zwischenkonfigurationen Ihrer Turingmaschine zum Debuggen einfach als Strings ausgeben. Definieren Sie auch eine Methode, welche die Startkonfiguration aus einem String erzeugt.

d) Definieren Sie eine Turingmaschine zur Multiplikation zweier Zahlen in unärer Darstellung (die Zahl 3 entspricht beispielsweise dem Wort `"111"`), welche auf dem Band durch ein Blank (`' '`) getrennt sind.

e) Analysieren Sie das Laufzeitverhalten Ihrer Turingmaschine für unterschiedlich große Eingaben.

f) Optimieren Sie Ihre Stackimplementierung so, daß sie die Arraygröße nicht nur um eins, sondern um eine Konstante `StackSpace` vergrößern bzw. verkleinern. Hierdurch muß das Array bei vielen push/pop-Operationen nicht mehr kopiert werden.

   Wie verändert sich das Laufzeitverhalten Ihrer Turingmaschine für unterschiedliche `StackSpace`-Werte?

g) Implementieren Sie eine "zeigerbasierte" Variante des Stacks. Definieren Sie hierfür zunächst eine Klasse `StackEntry`, welche einen Eintrag und einen Reststack aufnehmen kann. Kreieren Sie dann bei jeder push-Operation einen neuen Stackentry, dessen Wert bei der zugehörigen pop-Operation ausgelesen wird.

   Vergleichen Sie das Laufzeitverhalten der Turingmaschine unter Verwendung dieser Implementierung mit Ihrer Lösung für f).