Praktikum Compilerbau

Wintersemester 2005/06

Martin Hofmann, Andreas Abel, Hans-Wolfgang Loidl

Einführung

- Organisatorisches
- Aufgaben und Aufbau eines Compilers
- Überblick über das Praktikum
- Interpretation geradliniger Programme

Organisatorisches

- Das P richtet sich nach dem Buch *Modern Compiler Implementation in Java* von Andrew W Appel, CUP, 2005, 2. Aufl.
- Es wird ein Compiler für eine Teilmenge von Java: MiniJava entwickelt.
- Jede Woche wird ein Kapitel durchgenommen; ca. 30% VL und 70% Programmierung im Beisein der Dozenten.
- Die beaufsichtigte Programmierzeit wird i.A. nicht ausreichen; Sie müssen noch ca. 4h/Woche für selbstständiges Programmieren veranschlagen.
- Die Programmieraufgaben werden in Gruppen à zwei Teiln. bearbeitet (Extreme Programming).
- Scheinvergabe aufgrund erfolgreicher Abnahme des Programmierprojekts durch die Dozenten. Die Abnahme wird mündliche Fragen zum in der VL vermittelten Stoff enthalten.

Aufgaben eines Compilers

- Übersetzt Quellcode (in Form von ASCII Dateien) in Maschinensprache (“Assembler”)
- Lexikalische und Syntaxanalyse
- Semantische Analyse (Typüberprüfung und Sichtbarkeitsbereiche)
- Übersetzung in Zwischencode: keine lokalen Variablen, Sprunganweisungen und Funktionsaufrufe als einzige Kontrollstruktur
- Erzeugung von Maschineninstruktionen (architekturabhängig)
- Registerzuweisung
- Ausgabe in Binärformat

Zu verschiedenen Zeitpunkten können Optimierungen vorgenommen werden.

Geradlinige Programme

- Bestehen aus Zuweisungen, arithmetischen Ausdrücken, mehrstelligen Druckanweisungen.
- Beispiel:

  ```
  a := 5+3; b := (print(a, a-1), 10*a); print(b)
  ```

  Ausgabe:

  ```
  8 7
  80
  ```

  BNF Grammatik:

$$
\begin{array}{lll}
\mathit{Stm} & ::= & \mathit{Stm}\,\texttt{;}\,\mathit{Stm} \mid \mathit{ident}\,\texttt{:=}\,\mathit{Exp} \mid \texttt{print(}\,\mathit{ExpList}\,\texttt{)} \\
\mathit{Exp} & ::= & \mathit{ident} \mid \mathit{num} \mid \texttt{(}\,\mathit{Stm}\,\texttt{,}\,\mathit{Exp}\,\texttt{)} \mid \ldots \\
\mathit{ExpList} & ::= & \mathit{Exp} \mid \mathit{Exp}\,\texttt{,}\,\mathit{ExpList}
\end{array}
$$

Abstrakte Syntax in Java

```
abstract class Stm {}

class CompoundStm extends Stm {
   Stm stm1, stm2;
   CompoundStm(Stm s1, Stm s2) {stm1=s1; stm2=s2;}
}

class AssignStm extends Stm {
   String id; Exp exp;
   AssignStm(String i, Exp e) {id=i; exp=e;}
}

class PrintStm extends Stm {
   ExpList exps;
   PrintStm(ExpList e) {exps=e;}
}
```

Abstrakte Syntax in Java

```
abstract class Exp {}

class IdExp extends Exp {
   String id;
   IdExp(String i) {id=i;}
}

class NumExp extends Exp {
   int num;
   NumExp(int n) {num=n;}
}

class OpExp extends Exp {
   Exp left, right; int oper;
   final static int Plus=1,Minus=2,Times=3,Div=4;
   OpExp(Exp l, int o, Exp r) {left=l; oper=o; right=r;}
}
```

Abstrakte Syntax in Java

```
class EseqExp extends Exp {
   Stm stm; Exp exp;
   EseqExp(Stm s, Exp e) {stm=s; exp=e;}
}

abstract class ExpList {}

class PairExpList extends ExpList {
   Exp head; ExpList tail;
   public PairExpList(Exp h, ExpList t) {head=h; tail=t;}
}

class LastExpList extends ExpList {
   Exp head;
   public LastExpList(Exp h) {head=h;}
}
```

Beispiel”programm”

```
new CompoundStm(new AssignStm("a",new OpExp(new NumExp(5), OpExp.P
                                             new NumExp(3))),
 new CompoundStm(new AssignStm("b",
     new EseqExp(new PrintStm(new PairExpList(new IdExp("a"),
                          new LastExpList(new OpExp(new IdExp("a"), O
                                                    new NumExp(1)))))
                 new OpExp(new NumExp(10), OpExp.Times, new IdExp("a")
  new PrintStm(new LastExpList(new IdExp("b")))));
}
```

Programmieraufgabe für heute

- Implementieren einer Klasse `Table`, die Zuordnungen Bezeichner $\mapsto$ Werte, d.h. Umgebungen, modelliert.

  Umgebungen werden funktional implementiert, z.B. als Listen oder und können nicht imperativ verändert werden.

- Bereitstellen und Implementieren einer Methode

  ```
  Table interp(Table t)
  ```

  in der Klasse `Stm`.

  Der Aufruf `Table tneu = s.interp(t)` soll das Programm `s` in der Umgebung `t` auswerten und die daraus resultierende neue Umgebung in `tneu` abspeichern.

  Hierzu deklarieren Sie diese Methode in `Stm` als abstrakt und implementieren sie dann jeweils in den konkreten Unterklassen. Sie benötigen entsprechende Hilfsmethoden in der Klasse `Exp`.

- Viel Erfolg.