Compilerbau Vorlesung Wintersemester 2008, 09, 10, 11

Johannes Waldmann, HTWK Leipzig

24. Oktober 2011

Einleitung

Beispiel

Eingabe ($\approx$ Java):

```
{ int i;
  float prod;
  float [20] a;
  float [20] b;
  prod = 0;
  i = 1;
  do {
    prod = prod
      + a[i]*b[i];
    i = i+1;
  } while (i <= 20);
}
```

Ausgabe (Drei-Adress-Code):

```
L1: prod = 0
L3: i = 1
L4: t1 = i * 8
    t2 = a [ t1 ]
    t3 = i * 8
    t4 = b [ t3 ]
    t5 = t2 * t4
    prod = prod + t5
L6: i = i + 1
L5: if i <= 20 goto L4
L2:
```

Inhalt

- Motivation, Hintergründe
- lexikalische und syntaktische Analyse (Kombinator-Parser)
- syntaxgesteuerte Übersetzung (Attributgrammatiken)
- Code-Erzeugung ($+$ Optimierungen)
- statische Typsysteme
- Laufzeitumgebungen

Sprachverarbeitung

- mit Compiler:
  - Quellprogramm $\rightarrow$ Compiler $\rightarrow$ Zielprogramm
  - Eingaben $\rightarrow$ Zielprogramm $\rightarrow$ Ausgaben
- mit Interpreter:
  - Quellprogramm, Eingaben $\rightarrow$ Interpreter $\rightarrow$ Ausgaben
- Mischform:
  - Quellprogramm $\rightarrow$ Compiler $\rightarrow$ Zwischenprogramm
  - Zwischenprogramm, Eingaben $\rightarrow$ virtuelle Maschine $\rightarrow$ Ausgaben

Compiler und andere Werkzeuge

- Quellprogramm
- Präprozessor $\rightarrow$ modifiziertes Quellprogramm
- Compiler $\rightarrow$ Assemblerprogramm
- Assembler $\rightarrow$ verschieblicher Maschinencode
- Linker, Bibliotheken $\rightarrow$ ausführbares Maschinenprogramm

Phasen eines Compilers

- Zeichenstrom
- lexikalische Analyse $\rightarrow$ Tokenstrom
- syntaktische Analyse $\rightarrow$ Syntaxbaum
- semantische Analyse $\rightarrow$ annotierter Syntaxbaum
- Zwischencode-Erzeugung $\rightarrow$ Zwischencode
- maschinenunabhängige Optimierungen $\rightarrow$ Zwischencode
- Zielcode-Erzeugung $\rightarrow$ Zielcode
- maschinenabhängige Optimierungen $\rightarrow$ Zielcode

Methoden und Modelle

- lexikalische Analyse: reguläre Ausdrücke, endliche Automaten
- syntaktische Analyse: kontextfreie Grammatiken, Kellerautomaten
- semantische Analyse: Attributgrammatiken
- Code-Erzeugung: bei Registerzuordnung: Graphenfärbung

Anwendungen von Techniken des Compilerbaus

- Implementierung höherer Programmiersprachen
- architekturspezifische Optimierungen (Parallelisierung, Speicherhierarchien)
- Entwurf neuer Architekturen (RISC, spezielle Hardware)
- Programm-Übersetzungen (Binär-Übersetzer, Hardwaresynthese, Datenbankanfragesprachen)
- Software-Werkzeuge

Literatur

- Franklyn Turbak, David Gifford, Mark Sheldon: *Design Concepts in Programming Languages*, MIT Press, 2008.
  `http://cs.wellesley.edu/~fturbak/`

- Guy Steele, Gerald Sussman: *Lambda: The Ultimate Imperative*, MIT AI Lab Memo AIM-353, 1976

  (the original 'lambda papers', `http://library.readscheme.org/page1.html`)

- Alfred V. Aho, Monica S. Lam, Ravi Sethi and Jeffrey D. Ullman: *Compilers: Principles, Techniques, and Tools (2nd edition)* Addison-Wesley, 2007,
  `http://dragonbook.stanford.edu/`

Organisation

- pro Woche eine Vorlesung, eine Übung.

- Prüfungszulassung: regelmäßiges und erfolgreiches Bearbeiten von Übungsaufgaben

- Prüfung: Klausur (120 min, keine Hilfsmittel)

Beispiel: Interpreter (I)

arithmetische Ausdrücke:

```
data Exp = Const Integer
         | Plus Exp Exp | Times Exp Exp
    deriving ( Show )
ex1 :: Exp
ex1 = Times ( Plus ( Const 1 ) ( Const 2 ) )
value :: Exp -> Integer
value x = case x of
    Const i -> i
    Plus x y -> value x + value y
    Times x y -> value x * value y
```

Beispiel: Interpreter (II)

lokale Variablen und Umgebungen:

```
data Exp = ...
         | Let String Exp Exp | Ref String
ex2 :: Exp
ex2 = Let "x" ( Const 3 )
     ( Times ( Ref "x" ) (Ref "x" ) )
type Env = ( String -> Integer )
value :: Env -> Exp -> Integer
value env x = case x of
    Ref n -> env n
    Let n x b -> value ( \ m ->
      if n == m then value env x else env m )  b
    Const i -> i
    Plus x y -> value env x + value env y
    Times x y -> value env x * value env y
```

Übung (Haskell)

- Wiederholung Haskell
  - Interpreter/Compiler: ghci `http://haskell.org/`
  - Funktionsaufruf nicht `f(a,b,c+d)`, sondern `f a b (c+d)`
  - Konstruktor beginnt mit Großbuchstabe und ist auch eine Funktion
- Wiederholung funktionale Programmierung/Entwurfsmuster
  - rekursiver algebraischer Datentyp (ein Typ, mehrere Konstruktoren)
    (OO: Kompositum, ein Interface, mehrere Klassen)

– rekursive Funktion

- Wiederholung Pattern Matching:
  - beginnt mit `case ... of`, dann Zweige
  - jeder Zweig besteht aus Muster und Folge-Ausdruck
  - falls das Muster paßt, werden die Mustervariablen gebunden und der Folge-Ausdruck auswertet

Übung (Interpreter)

- Benutzung:
  - Beispiel für die Verdeckung von Namen bei geschachtelten Let
  - Beispiel dafür, daß der definierte Name während seiner Definition nicht sichtbar ist
- Erweiterung:

  Verzweigungen mit C-ähnlicher Semantik:

  Bedingung ist arithmetischer Ausdruck, verwende 0 als Falsch und alles andere als Wahr.

```
data Exp = ...
         | If Exp Exp Exp
```

Umgebungen

Umgebung ist (partielle) Funktion von Name nach Wert

Realisierungen: `type Env = String -> Int`

`import Data.Map ; type Env = Map String Int`

Operationen:

- `empty :: Env` leere Umgebung
- `lookup :: Env -> String -> Int`

  Notation: $e(x)$
- `extend :: Env -> String -> Int -> Env`

  Notation: $e[x/v]$

Spezifikation:

- $e[x/v](x) = v, \quad x \neq y \Rightarrow e[x/v](y) = e(y)$

Inferenz-Systeme

Motivation

- inferieren $=$ ableiten
- Inferenzsystem I, Objekt O,

  Eigenschaft $I \vdash O$ (in I gibt es eine Ableitung für O)
- damit ist I eine *Spezifikation* einer Menge von Objekten
- man ignoriert die *Implementierung* ($=$ das Finden von Ableitungen)
- Anwendungen im Compilerbau:

  Auswertung von Programmen, Typisierung von Programmen

Definition

ein *Inferenz-System* I besteht aus

- Regeln (besteht aus Prämissen, Konklusion)

  Schreibweise $\frac{P_1,\ldots,P_n}{K}$

- Axiomen ($=$ Regeln ohne Prämissen)

eine *Ableitung* für F bzgl. I ist ein Baum:

- jeder Knoten ist mit einer Formel beschriftet

- jeder Knoten (mit Vorgängern) entspricht Regel von I

- Wurzel ist mit F beschriftet

Schreibweise: $I \vdash F$

Inferenz-Systeme (Beispiel 1)

- Grundbereich $=$ Zahlenpaare $\mathbb{Z} \times \mathbb{Z}$
- Axiom:

$$\frac{}{(13, 5)}$$

- Regel-Schemata:

$$\frac{(x, y)}{(x - y, y)}, \quad \frac{(x, y)}{(x, y - x)}$$

kann man $(1, 1)$ ableiten? $\quad (-1, 5)$? $(2, 4)$?

Inferenz-Systeme (Beispiel 2)

- Grundbereich: Zeichenketten aus $\{0,1\}^*$
- Axiom:

$$\frac{}{01}$$

- Regel-Schemata (für jedes u, v):

$$\frac{0u, v0}{u1v}, \quad \frac{1u, v1}{u0v}, \quad \frac{u}{\mathrm{reverse}(u)}$$

Leite 11001 ab. Wieviele Wörter der Länge k sind ableitbar?

Inferenz-Systeme (Beispiel 3)

- Grundbereich: endliche Folgen von ganzen Zahlen
- Axiome: jede konstante Folge (Bsp. $[3, 3, 3, 3]$)
- Schlußregeln:
  - swap$_k$: $\dfrac{[\ldots, x_k, x_{k+1}, \ldots]}{[\ldots, x_{k+1} + 1, x_k - 1, \ldots]}$
  - rotate: $\dfrac{[x_1, \ldots, x_n]}{[x_2, \ldots, x_n, x_1]}$

Aufgaben:

- Ableitungen für $[5, 3, 1, 3], [7, 7, 1]$
- jede Folge der Form $[z, 0, \ldots, 0]$ ist ableitbar
- $[5, 3, 3]$ ist nicht ableitbar

Inferenz von Werten

- Grundbereich: Aussagen der Form $\mathsf{wert}(p, z)$ mit $p \in \texttt{Exp}$, $z \in \mathbb{Z}$

```
data Exp = Const Integer
         | Plus Exp Exp
         | Times Exp Exp
```

- Axiome: $\mathsf{wert}(\texttt{Const}\, z, z)$
- Regeln:

$$\frac{\mathsf{wert}(X, a), \mathsf{wert}(Y, b)}{\mathsf{wert}(\texttt{Plus}\ X\ Y, a + b)}, \quad \frac{\mathsf{wert}(X, a), \mathsf{wert}(Y, b)}{\mathsf{wert}(\texttt{Times}\ X\ Y, a \cdot b)}, \ldots$$

Umgebungen

- Grundbereich: Aussagen der Form $\mathsf{wert}(E, p, z)$

  (in Umgebung E hat Programm p den Wert z)

  Umgebungen konstruiert aus $\emptyset$ und $E[v := p]$

- Regeln für Operatoren $\dfrac{\mathsf{wert}(E, X, a), \mathsf{wert}(E, Y, b)}{\mathsf{wert}(E, \mathtt{Plus}XY, a + b)}, \ldots$

- Regeln für Umgebungen

$$\frac{}{\mathsf{wert}(E[v := b], v, b)}, \quad \frac{\mathsf{wert}(E, v', b')}{\mathsf{wert}(E[v := b], v', b')} \text{ für } v \neq v'$$

- Regeln für Bindung: $\dfrac{\mathsf{wert}(E, X, b), \mathsf{wert}(E[v := b], Y, c)}{\mathsf{wert}(E, \mathtt{let}\ v = X\ \mathtt{in}\ Y, c)}$

Aussagenlogische Resolution

Formel $(A \vee \neg B \vee \neg C) \wedge (C \vee D)$ in konjunktiver Normalform dargestellt als $\{\{A, \neg B, \neg C\}, \{C, D\}\}$
(Formel $=$ Menge von Klauseln, Klausel $=$ Menge von Literalen, Literal $=$ Variable oder negierte Variable)

folgendes Inferenzsystem heißt *Resolution*:

- Axiome: Klauselmenge einer Formel,
- Regel:
  - Prämissen: Klauseln K_1, K_2 mit $v \in K_1, \neg v \in K_2$
  - Konklusion: $(K_1 \setminus \{v\}) \cup (K_2 \setminus \{\neg v\})$

Resolution (Eigenschaften)

die Formel (Klauselmenge) ist nicht erfüllbar $\iff$ *die leere Klausel ist durch Resolution ableitbar.*

Beweispläne:

- $\Rightarrow$: Gegeben ist die nicht erfüllbare Formel. Gesucht ist eine Ableitung für die leere Klausel. Methode: Induktion nach Anzahl der in der Formel vorkommenden Variablen.

- $\Leftarrow$: Gegeben ist die Ableitung der leeren Klausel. Zu zeigen ist die Nichterfüllbarkeit der Formel. Methode: Induktion nach Höhe des Ableitungsbaumes.