

Software-Reengineering

WS 2006/07
G. Vogel

2. Übungsblatt

Dieses Übungsblatt soll den Vorlesungsstoff zur quellennahen Analyse mittels angepasster Compilerbau-Technologie wiederholen und vertiefen. Das Übungsblatt wird am 7./8. November besprochen.

Aufgabe 2.1 *Programmrepräsentation/Ausführungssemantik*

Die Aufgabe 1.5 auf dem ersten Übungsblatt hat sich mit der Semantik der Case-Anweisungen von Ada beschäftigt.

```
case X is
  when 1      => Do_Something;
  when 2..3|5 => Do_Something_Else;
  when others => Do_Nothing;
end case;
```

- Zeichnen Sie einen abstrakten Syntaxbaum für die Case-Anweisung aus dem Beispiel und für ein semantisch äquivalentes Programm, das If-Anweisungen verwendet.
- Welchen Vorteil hat die zweite Art der Darstellung für die Programm-analyse? Durch welche zusätzlichen Vorkehrungen kann gleichzeitig die Quellennähe der Repräsentation gewahrt werden? Zeichnen Sie für das Beispiel die entsprechende Zwischendarstellung.

Aufgabe 2.2 *Abstrakte Syntax*

Gegeben ist die folgende EBNF-Grammatik für einfache Ausdrücke (Lexeme sind in Anführungszeichen gesetzt). Für die Ausdrücke soll im folgenden ein Wert berechnet werden. Zur Auswertung und Interpretation werden die Ausdrücke in Form von abstrakten Syntaxbäumen dargestellt. Bei gleichen Operator-Präzedenzen soll von links nach rechts ausgewertet werden.

Beispiel: $5 * 6 / 2 = (5 * 6) / 2$ oder $3 - 2 - 1 = (3 - 2) - 1$

```
expr      ::= simple_expr {relop simple_expr} .
relop     ::= "=" | "<" | "<=" | ">=" | "<" | ">" .
simple_expr ::= term {addop term} .
addop     ::= "+" | "-" | "or" .
term      ::= factor {mulop factor} .
mulop     ::= "*" | "/" | "mod" | "and" .
factor    ::= num
           | "(" expr ")"
```

```

num      ::= | "not" factor .
digit    ::= ["-"] digit {digit} [ "." digit {digit} ] .
id       ::= letter {letter} .
letter   ::= "a" | "b" | "c" | ... | "z" .

```

- Wie könnte ein abstrakter Syntaxbaum aussehen? Beschreiben Sie die Knoten und deren Attribute, beispielsweise in Form von Klassendiagrammen. Die Kanten zwischen den Knoten sollen dabei über Attribute dargestellt werden.
- Erstellen Sie eine Beschreibung der Klassen in Pseudo-Code oder einer Programmiersprache ihrer Wahl und beschreiben Sie einen Algorithmus zur Berechnung des Wertes von Ausdrücken. Die Werte der Variablen (id) sollen dabei zur Zeit der Auswertung aus einer Tabelle mit entsprechenden Getter-Methoden ausgelesen werden.
- Geben Sie einen abstrakten Syntaxbaum für den folgenden Ausdruck an:

$$a + (b + c) - 5 * 10 \bmod 3$$
- Eine einfache Optimierung würde konstante Werte von Teilausdrücken vorberechnen, sodass diese nur ein Mal für verschiedene Belegungen der Variablen bestimmt werden müssen. Beschreiben Sie, wie konstante Teilausdrücke erkannt werden können und, wie eine Transformation auf dem abstrakten Syntaxbaum zur Optimierung durchgeführt werden könnte.

Aufgabe 2.3 *Kontrollfluss*

- Geben Sie für das folgende C-Programm eine Repräsentation sowohl des intra- als auch des interprozeduralen Kontrollflusses an. Beachten Sie die besondere Semantik des `&&`-Operators: Im Falle von `1&&r` wird `r` nur dann evaluiert, wenn `1` den Wert `true` hat.
- Bestimmen Sie für die intraprozeduralen Flussgraphen der beiden Funktionen die Dominanz- und Postdominanzbäume und die Kontrollabhängigkeiten.

```

1  int sum, prod;
2  int debug = 1;
3  void foo (int value) {
4      if ((value > 0) && debug) {
5          sum = sum + value;
6          prod = prod * value;
7          foo (value - 1);
8      }
9  }
10 int main () {
11     sum = 0;
12     prod = 1;
13     foo (10);
14     return 0;
15 }

```