Universität Karlsruhe (TH)

Lehrstuhl für Programmierparadigmen

Sprachtechnologie und Compiler WS 2008/2009 — http://pp.info.uni-karlsruhe.de/

Dozent: Prof. Dr.-Ing. G. Snelting — snelting@ipd.info.uni-karlsruhe.de

Übungsleiter: Matthias Braun — braun@ipd.info.uni-karlsruhe.de

Übungsblatt 11 — Ausgabe: 19.1.2008 — Besprechung: 21.1.2009

Aufgabe 1: *Trampolines (Zusatzaufgabe)*

Funktionszeiger sind laut C Sprachstandard ganz normale Zeiger (man kann Sie zu **void**∗ und zurück casten). Der gcc Compiler unterstützt geschachtelte Funktionen und kann diese auch beliebigen Funktionszeigern zuweisen. Es wird also anscheinend kein speziellere Datentyp für Closures angelegt, der einen Funktionszeiger und einen Zeiger auf die Umgebung enthält. Wie kann das Funktionieren?

Aufgabe 2: *x86-Assembler*

Gegeben sei die folgende 3-Adresszwischensprache:

t_i : adr *identifier*	t_i := Speicheradresse der Variablen *identifier*
t_i : load t_j	t_i := Memory [t_j]
t_i : const *int_const*	t_i := *int_const*
t_i : plus t_j t_k	t_i := t_j + t_k
t_i : less t_j t_k	t_i := t_j < t_k
t_i : if t_j t_k	Wenn t_j <> 0 springe Anweisung t_k sonst führe t_{i+1} aus
t_i : goto t_j	Springe zur Anweisung t_j
t_i : store t_j t_k	Memory [t_j] := t_k
t_i : arg t_j	t_j als Argument für den folgenden Call benutzen
t_i : call *foo*	Unterprogramm *foo* aufrufen

Gegeben Sei folgendes Zwischensprachprogramm:

```
t0:   adr     i
t1:   const   1
t2:   store   t0  t1

t3:   adr     i
t4:   load    t3
t5:   const   1
t6:   plus    t4  t5
t7:   adr     i
t8:   store   t7  t6

t9:   adr     j
t10:  load    t9
t11:  adr     k
t12:  load    t11
t13:  plus    t10  t12
t14:  adr     i
t15:  store   t14

t16:  adr     i
t17:  load    t16
t18:  adr     j
t19:  load    t18
t20:  less    t17  t19

t21:  if      t20  t27
```

```
t22: adr     j
t23: load    t22
t24: adr     min
t25: store   t24
t26: goto    31
t27: adr     i
t28: load    t27
t29: adr     min
t30: store   t29
t31:
```

2.1 *Quellprogramm*

Wie könnte die C-Funktion aussehen, aus der der Zwischencode erzeugt wurde?

2.2 *1:1 Umsetzung*

Übersetzen Sie den Zwischencode in IA32-Assembler. Ersetzen Sie zunächst jeden Befehl durch einen oder mehrere x86 Befehle.

2.3 *Optimierung*

Geben Sie besseren IA32-Code an.

Aufgabe 3: *Zwischensprachen*

3.1 *Grundblagen*

- In welche Einheiten wird ein Programm in der Zwischensprache typischerweise eingeteilt?
- Wie ist ein Grundblock definiert?
- Wie ist ein erweiterter Grundblock definiert?
- Wofür sind erweiterte Grundblöcke nützlich?

3.2 *Übersetzung*

```
void f(void)
{
    int x = rand();

    x = rand();
label:
    x = x + i;
    if (i > 100) {
        goto label;
    } else if (x < 1) {
        return 1;
    }

    int i = 10;
    do {
        if (i < 12)
            f();
        else
            i = i + 2;

        i = i + 1;
    } while(i < 20);
}
```

- Übersetzen Sie das folgende Programm in die Zwischensprache aus der ersten Aufgabe!
- Markieren Sie alle Grundblöcke!
- Teilen Sie das Programm in erweiterte Grundblöcke ein!

Aufgabe 4: *Zwischensprachen 2*

Gegeben sei folgende C-Funktion f:

```
void f(void)
{
    int a = 4;
    int b = 42;

    while (1) {
        int t = a;
        a = b;
        b = t;
        printf("%d %d\n", a, b);
    }
}
```

4.1 *Dynamisches Verhalten*

Was gibt die Funktion aus?

4.2 *Tripel-Form*

Wie könnte f in einer Zwischensprache mit Tripel-Form aussehen?

4.3 *Triple-Form mit SSA-Eigenschaft*

Wie könnte f in einer Zwischensprache mit Tripel-Form und SSA-Eigenschaft aussehen?

4.4 *Semantik der Φ-Funktion*

Welche besondere Semantik haben Φ-Funktionen? Worauf muss man insbesondere bei diesem Beispiel achten?