Universität Karlsruhe (TH)

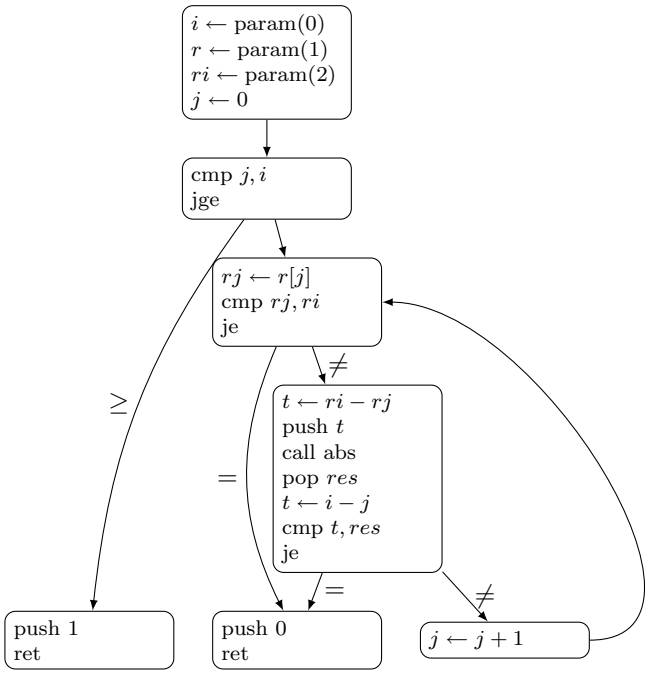
Lehrstuhl für Programmierparadigmen

Sprachtechnologie und Compiler II SS 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

Lösung zu Übungsblatt 7 | Ausgabe: 14.7.2009 | Besprechung: 21.7.2009

Aufgabe 1: *Linear Scan Registerzuteilung*

Auf einem Programm in Tripelform soll eine linear Scan Registerzuteilung durchgeführt werden. Ordnen Sie dazu die Grundblöcke zunächst in einer (beliebigen) Post-Order Anordnung. Zeichnen Sie die Lebendigkeitsintervalle der Variablen für diese Anordnung und teilen Sie für das Programm nach dem Linear-Scan Verfahren Register zu. Gehen Sie davon aus, dass 4 physische Register vorhanden sind. Es genügt ein naives Auslagerungsverfahren zu benutzen: Unmittelbar nach jeder Definition von x wird $m_x \leftarrow \text{spill}(x)$ eingefügt, vor jeder Benutzung ein $x \leftarrow \text{reload}(m_x)$.



Lösung:

1: $i \leftarrow \text{param}(0)$
2: $r \leftarrow \text{param}(1)$
3: $ri \leftarrow \text{param}(2)$
4: $j \leftarrow 0$
5: cmp j, i
6: jge L4
7: L1 :
8: $rj \leftarrow r[j]$
9: cmp rj, ri
10: je L3
11: L2 :
12: $t \leftarrow ri - rj$
13: push t
14: call abs
15: pop res
16: $t \leftarrow i - j$
17: cmp t, res
18: je L3
19: $j \leftarrow j + 1$
20: jmp L2
21: L3 :
22: push 0
23: ret
24: L4 :
25: push 1
26: ret

Lebendigkeitsintervalle:
i: 1-20
r: 2-20
ri: 3-20
j: 4-20
rj: 8-12
t: 12-17
res: 15-17
Registerzuteilung:

1. $i \leftarrow r_0$
2. $r \leftarrow r_1$
3. $ri \leftarrow r_2$
4. $j \leftarrow r_3$
5. j wird ausgelagert; $rj \leftarrow r_3$
6. Register r_3 freigeben; $t \leftarrow r_3$
7. ri wird ausgelagert; $res \leftarrow r_2$

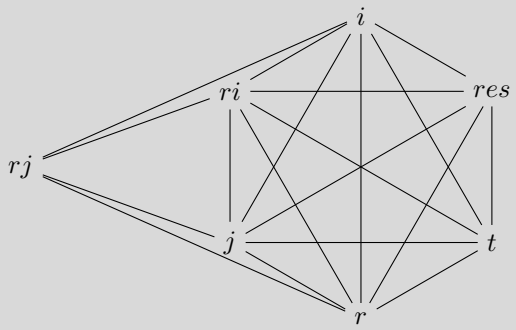
Aufgabe 2: *Graph-Färben nach Chaitin*

Erstellen Sie für das Programm aus Aufgabe 1 einen Interferenzgraphen und führen Sie eine Registerzuteilung nach dem Chaitin Verfahren für eine Architektur mit 4 Registern durch.

Lösung:

Hinweis: Wir benutzen das „optimistic coloring" Technik von Briggs et. al. [a].

Interferenzgraph



Auslagerungskosten

Die Kosten ergeben sich aus der Summe der benötigten Spill und Reload Befehle gewichtet mit $c*10^d$, wobei c die architekturabhängigen Kosten für den jeweiligen Befehl sind und d die Schleifentiefe. Mit $c := 1$ ergeben sich folgende Auslagerungskosten:

Knoten	Kosten
r	$1 + 10$
i	$1 + 11$
res	$10 + 10$
ri	$1 + 20$
rj	$10 + 20$
j	$11 + 21$
t	$20 + 20$

Färben und Auslagern

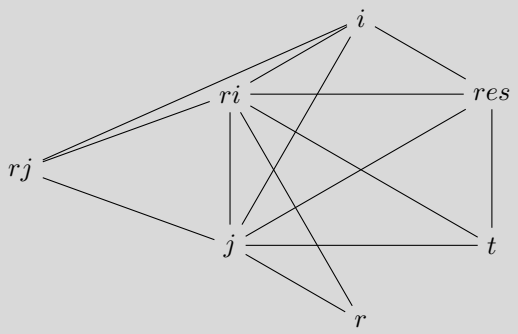
Auf den Stack gelegt werden:
r, rj, i, res, ri, j, t dabei sind r und i potentielle Spills.
Zugewiesene Farben:

Variable	Register
t	r_0
j	r_1
ri	r_2
res	r_3
i	spilled
rj	r_0
r	spilled

[a] Preston Briggs, Keith D. Cooper and Linda Torczon - *Improvements to Graph Coloring Register Allocation*

Spillcode einfügen, neuen Interferenzgraph erzeugen

```
 1: i ← param(0)
 2: m_i ← spill(i)
 3: r ← param(1)
 4: m_r ← spill(r)
 5: ri ← param(2)
 6: j ← 0
 7: i ← reload(m_i)
 8: cmp j, i
 9: jge L4
10: L1 :
11: r ← reload(m_r)
12: rj ← r[j]
13: cmp rj, ri
14: je L3
15: L2 :
16: t ← ri − rj
17: push t
18: call abs
19: pop res
20: i ← reload(m_i)
21: t ← i − j
22: cmp t, res
23: je L3
24: j ← j + 1
25: jmp L2
26: L3 :
27: push 0
28: ret
29: L4 :
30: push 1
31: ret
```



Färben und Auslagern

Auf den Stack gelegt werden:
r, i, rj, j, t, res, ri
Zugewiesene Farben:

Variable	Register
r	r_0
i	r_2
rj	r_1
j	r_3
t	r_2
res	r_1
ri	r_0

Ergebnis der Registerallokation

1: $r_2 \leftarrow \text{param}(0)$
2: $m_{r_2} \leftarrow \text{spill}(r_2)$
3: $r_0 \leftarrow \text{param}(1)$
4: $m_{r_0} \leftarrow \text{spill}(r_0)$
5: $r_0 \leftarrow \text{param}(2)$
6: $r_3 \leftarrow 0$
7: $r_2 \leftarrow \text{reload}(m_{r_2})$
8: cmp r_3, r_2
9: jge L4
10: L1 :
11: $r_0 \leftarrow \text{reload}(m_{r_0})$
12: $r_1 \leftarrow r_0[r_3]$
13: cmp r_1, r_0
14: je L3
15: L2 :
16: $r_2 \leftarrow r_0 - r_1$
17: push r_2
18: call abs
19: pop r_1
20: $r_2 \leftarrow \text{reload}(m_{r_2})$
21: $r_2 \leftarrow r_2 - r_3$
22: cmp r_2, r_1
23: je L3
24: $r_3 \leftarrow r_3 + 1$
25: jmp L2
26: L3 :
27: push 0
28: ret
29: L4 :
30: push 1
31: ret

Aufgabe 3: *SSA-Registerzuteilung*

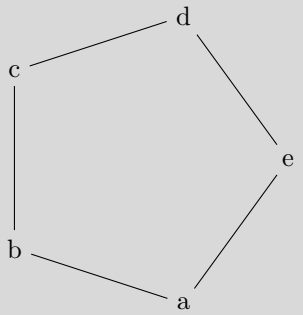
3.1 C_5

Geben Sie ein Programm an, das den C_5 als Interferenzgraphen hat, ohne jedoch eine Variable innerhalb eines Grundblocks mehrfach zu definieren.

Lösung:

```
a = ...
if(...)
 b = ...
 c = a + b
 d = b + c
 e = c + d
 print(d)
else
 e = ...
 print(a)
print(e)
```

Führt zu IG:



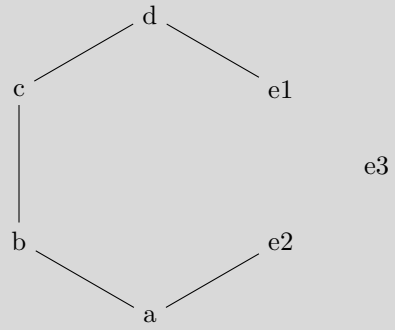
3.2 *SSA*

Überführen Sie ihr Programm in SSA-Form. Geben Sie eine PEO und eine Färbung nach Hack/Goos an.

Lösung:

```
a = ...
if(...)
 b = ...
 c = a + b
 d = b + c
 e1 = c + d
 print(d)
else
 e2 = 42
 print(a)
e3 = phi(e1, e2)
print(e3)
```

Führt zu IG:

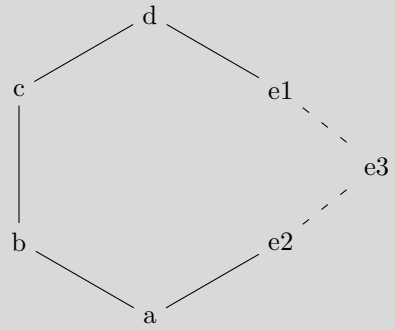


PEO: e3, e2, e1, d, c, b, a
Färbung: a = 1, b = 2, c = 1, d = 2, e1 = 1, e2 = 2, e3 = 1

3.3 *Zuteilung/Verschmelzen*

Tragen Sie die Affinitätskanten in den Interferenzgraphen ein. Bestimmen Sie die optimale Zuteilung bei 2 und bei 3 verfügbaren Registern.

Lösung:



Färbung mit 3 Registern: a = 1, b = 2, c = 1, d = 2, e1 = 3, e2 = 3, e3 = 3
Färbung mit 2 Registern: a = 1, b = 2, c = 1, d = 2, e1 = 1, e2 = 2, e3 = 1

3.4 *SSA-Registerzuteilung - Theorie*

Minimales Färben ist im allgemeinen ein NP-schweres Problem. Zu jedem (Interferenz-)Graphen lässt sich ein passendes Programm (mit einer grösse linear in der Anzahl der Kanten) angeben. Ein Programm lässt sich in polynomieller Zeit in SSA-Form und zurück transformieren. Interferenzgraphen von SSA-Programmen sind

chordal und lassen sich in linearer Zeit färben. Warum lassen sich hiermit keine minimalen Färbungen für allgemeine (Interferenz-)Graphen erzeugen?

Lösung:

Die Transformation eines Programmes in SSA-Form und zurück resultiert in einem Programm, dass das gleiche beobachtbare Verhalten besitzt aber nicht das gleiche Programm ist. Zum Beispiel können durch den ϕ-Funktionen zusätzliche Kopien entstanden sein. Somit lässt sich nicht garantieren, dass die Lösung als Färbung für das ursprüngliche Programm geeignet ist.