

Lecture 21

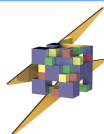
Register Allocation

이재진

서울대학교 데이터사이언스대학원

서울대학교 공과대학 컴퓨터공학부

<http://aces.snu.ac.kr/~jlee>



Register Allocation

- Deciding what values to hold in what registers
- Register allocation
 - Select the set of variables that will reside in registers at each point in the program
- Register assignment
 - Pick the specific register that a variable will reside in
- Finding an optimal assignment of registers to variables is NP-complete
- The order in which computations are performed can affect the efficiency of the target code
 - Some computation orders require fewer registers to hold intermediate result
 - NP-complete



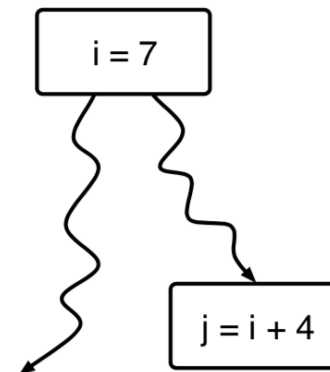
Chaitin's Algorithm

- The first register allocation algorithm that made use of coloring of the interference graph for both register allocation and spilling



Live Variables

- A variable x is live at a point p if the value of x at p could be used along some path in the flow graph starting at p
- Used in
 - Register allocation
 - Code motion in loops
 - Elimination of useless assignments (dead code elimination)
- Detect live variables using a data-flow analysis technique
 - Data flows backwards



Interference Graph

- Interference
 - A condition that prevents variables a and b from being allocated to the same register
- The nodes in an interference graph represent temporaries or symbolic registers

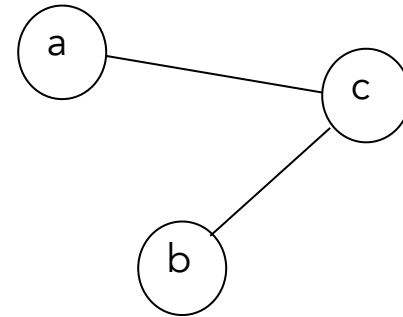
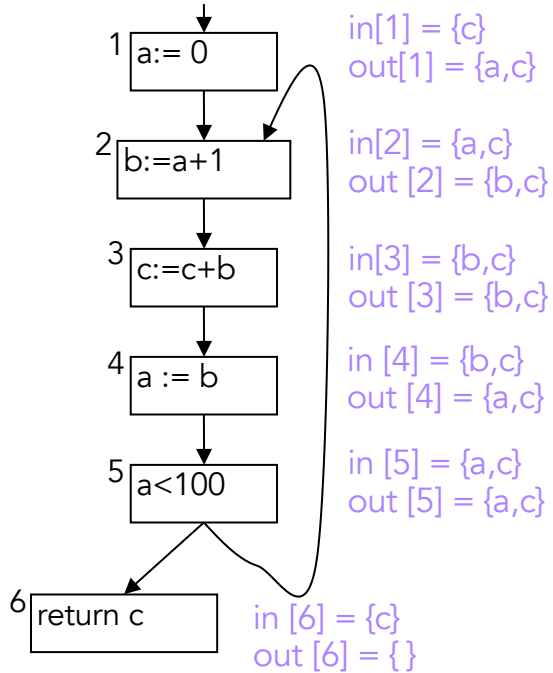


Interference Graph (cont'd)

- An edge connects the nodes for a and b iff there is an interference between a and b
 - A variable is live if it holds a value that may be needed in the future
 - At any non-move instruction that defines a in a block n , add interference edges (a, b) for all $b \in out[n]$
 - At any move instruction MOVE a, c (e.g., $a := c$) in a block n , add interference edges (a, b) for all $b \in (out[n] - \{c\})$

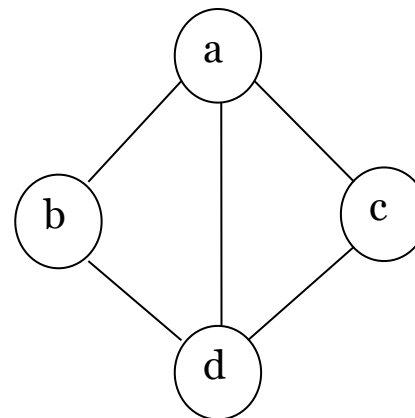
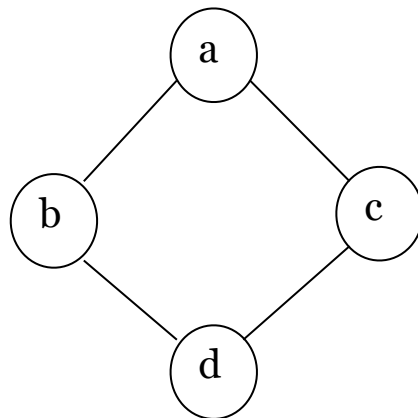


Interference Graph (cont'd)



Graph Coloring

- Each node in the interference graph represents a temporary. Each edge indicates a pair of temporaries cannot be assigned to the same register
- Color the interference graph using as few colors as possible, but no pair of nodes connected by an edge may be assigned the same color



Graph Coloring (cont'd)

- If our target machine has K registers, and we can K -color the graph, then the coloring is a valid register assignment
- If there is no K -coloring, some of variables and temporaries should be kept in memory (spilling)
- Register allocation is an NP-complete problem
- We use a linear approximation algorithm



Coloring by Simplification

- Assume that our machine has K registers
- (Build) Construct the interference graph (G)
- (Simplify) Repeatedly remove nodes of degree less than K and push it on a stack. Suppose G contains a node m with fewer than K neighbors. If $G - \{m\}$ can be K -colorable, so can G
- (Spill) If the simplification step produces a graph that has nodes only of significant degree (nodes of degree $\geq K$), mark some node for spilling and push it on the stack. Continue simplification
- (Select) Rebuild the original graph by repeatedly adding a node from the top of the stack and select a color for the node. If it is not a potential spill node, there is always a color for the node. If the node is a potential spill node and there is no color available, it becomes an actual spill
- (Start over) If there were any actual spills, rewrite the code and goto step 1



At the Beginning

$t2 = t1 - 1$

$t4 = t3$

$t6 = t5$

L1: $t2 = t2 + 1$

$t4 = t4 - 1$

$t7 = M[SP-8]$

$t8 = M[SP-12]$

if ($t7 \leq t8$) goto L2

$t6 = t7$

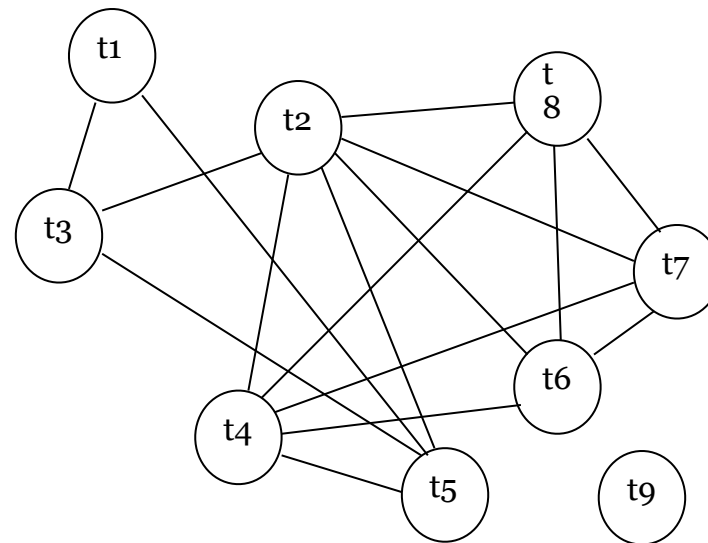
L2: $t8 = t2$

if ($t4 \geq 0$) goto L1

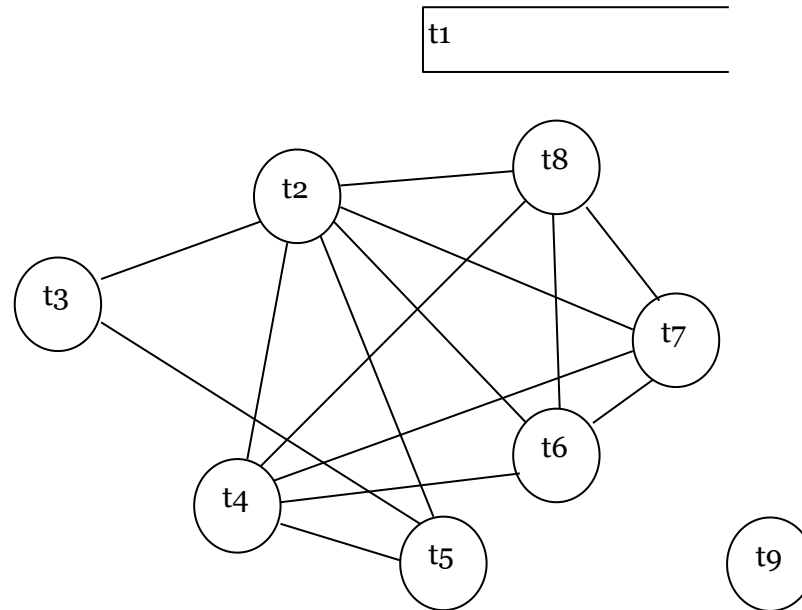
$t9 = t6 + t8$

return $t9$

Available machine registers: r0, r1, r2, and r3

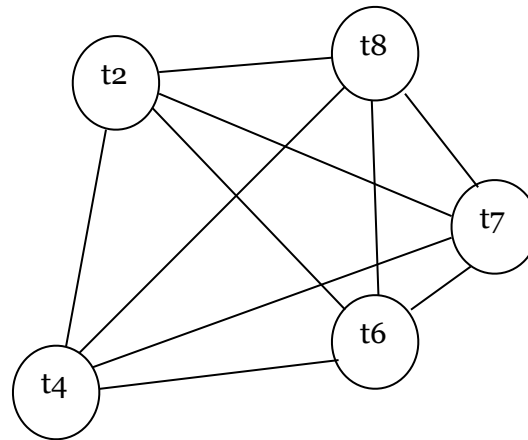


Simplification

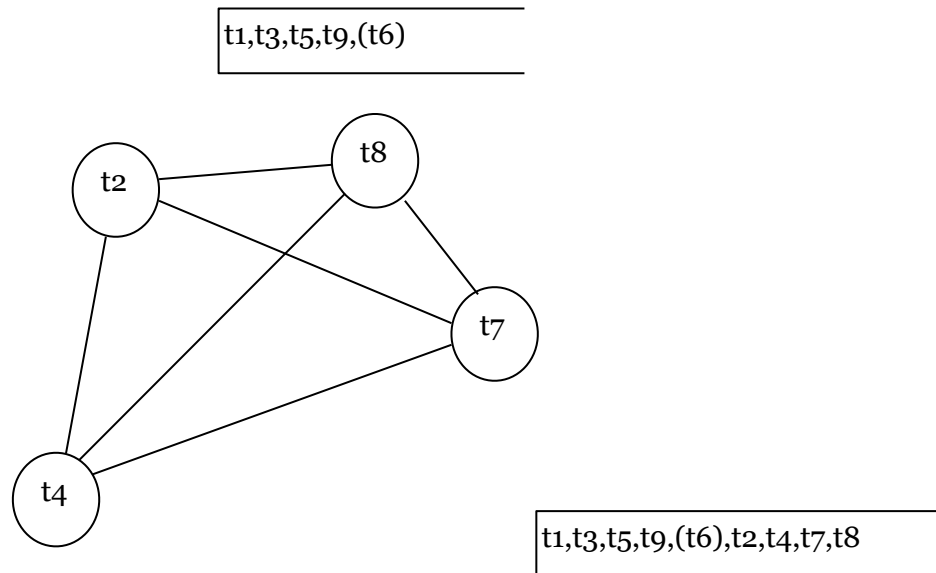


Simplification (cont'd)

t1,t3,t5,t9



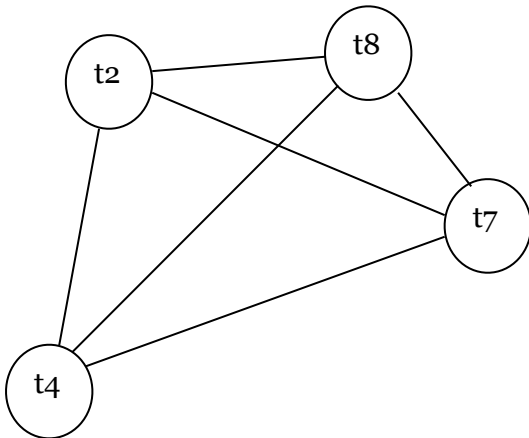
Spilling and Simplification



Selection and Rewriting the code

t8: r0
t7: r1
t4: r2
t2: r3

t6 is an actual spill



t2 = t1 - 1

t4 = t3

t6 = t5

M[SP-16] = t6

L1: t2 = t2 + 1

t4 = t4 - 1

t7 = M[SP-8]

t8 = M[SP-12]

if (t7 <= t8) goto L2

t6 = t7

M[SP-16] = t6

L2: t8 = t2

if (t4 >= 0) goto L1

t10 = M[SP-16]

t9 = **t10** + t8

return t9

t2 = t1 - 1

t4 = t3

t6 = t5

L1: t2 = t2 + 1

t4 = t4 - 1

t7 = M[SP-8]

t8 = M[SP-12]

if (t7 <= t8) goto L2

t6 = t7

L2: t8 = t2

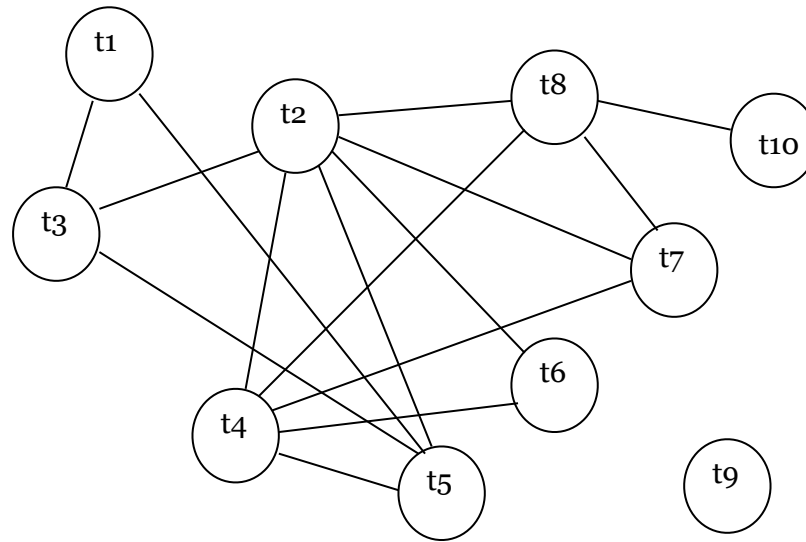
if (t4 >= 0) goto L1

t9 = t6+t8

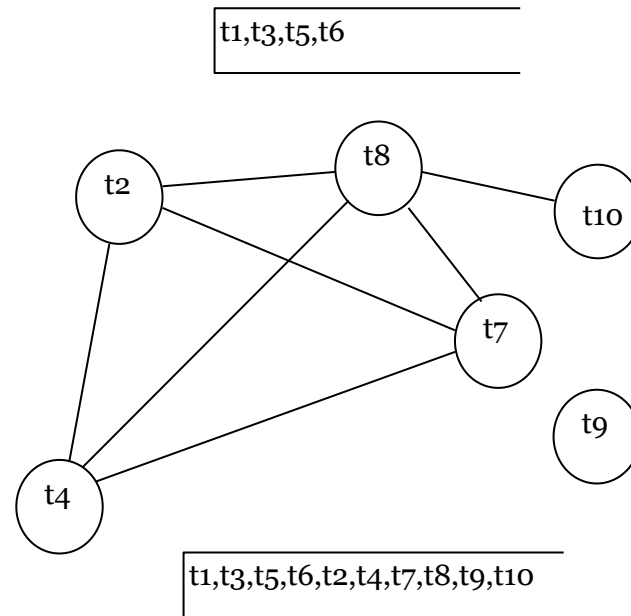
return t9



Building a New Interference Graph



Simplification



Selection

- t10: r0
- t9: r0
- t8: r1
- t7: r2
- t4: r3
- t2: r0
- t6: r1
- t5: r1
- t3: r2
- t1: r3

