



Sequence 5.4 – Register Allocation

P. de Oliveira Castro S. Tardieu

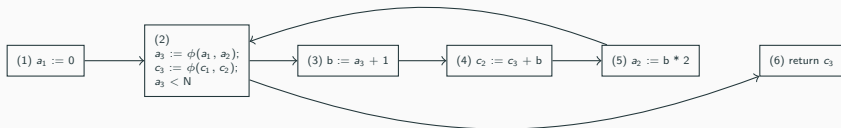
Interference Graph

```
function f(c : int) =  
  let var a := 0  
  in  
    while (a < N) do  
      let var b := a + 1 in  
        c := c + b;  
        a := b * 2  
      end;  
    c  
  end
```

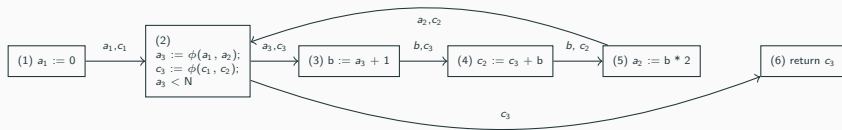
- Variables are stored either in the memory or in a register.
- Register accesses are much faster than memory accesses.
- LLVM uses an unlimited number of values, but physical registers are limited. How can we map LLVM values to a reduced number of physical registers?

CFG in SSA Form

```
function f(c : int) =  
  let var a := 0  
  in  
    while (a < N) do  
      let var b := a + 1 in  
        c := c + b;  
        a := b * 2  
      end;  
    c  
  end
```

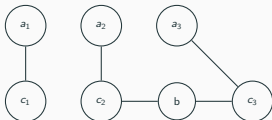


Liveness Analysis



■ Liveness Analysis

- b is live between 3 and 5
 - c_1 is live between 1 and 2; c_2 is live between 4 and 2
 - a_1 is live between 1 and 2; a_2 is live between 5 and 2
 - ...
- *Interference graph*: two nodes are connected if they can both be alive at the same time



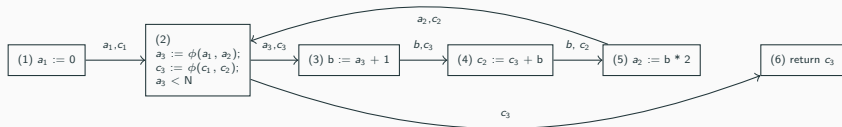
Flow Equations

$$in(N) = use(N) \cup (out(N) - def(N))$$

$$out(N) = \bigcup_{S \in succ(N)} in(S)$$

- Values used in a node must be live in the inputs
- Values live in the outputs are either live in the inputs or defined in the node
- Values live in the outputs must be live in the inputs of the successor nodes

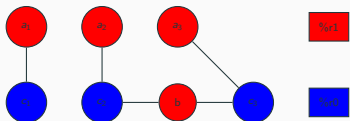
Flow Equations: an Example



- $in(6) = use(6) = \{c_3\}$
- $out(2) = in(6) = \{c_3\}$
- $in(2) = (out(2) - def(2)) \cup use(2) = \{\phi(c_1, c_2), \phi(a_1, a_2)\}$
- $out(1) = in(2) = \{a_1, c_1\}$ (we resolve ϕ nodes)
- $out(5) = in(2) = \{a_2, c_2\}$ (we resolve ϕ nodes)
- $in(5) = (out(5) - def(5)) \cup use(5) = \{b, c_2\}$
- $out(4) = in(5) = \{b, c_2\}$
- ... until fixed point is reached!

- Why is there always a fixed point?
- Why is a reversed propagation more efficient?
- Flow analysis is a versatile framework to implement many optimizations

Register Allocation = Graph Coloring

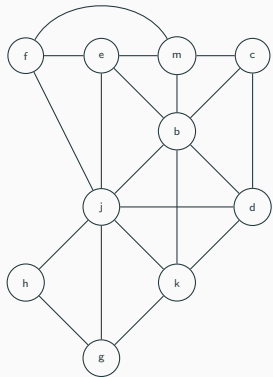


```
mov r1, #0
L1: add r1, r1, #1
    add r0, r0, r1
    mul r1, r1, #2
    cmp r1, #N
    blt L1
bx lr
```


Graph Coloring

- If k physical registers are available, the graph allocation problem is equivalent to coloring the interference graph with k colors or less.
- NP-complete problem
- ... but good heuristic: coloring by simplification.

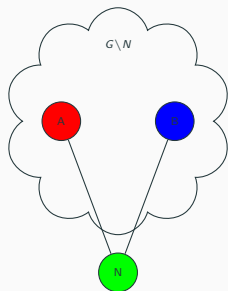
Coloring by Simplification



Simplification

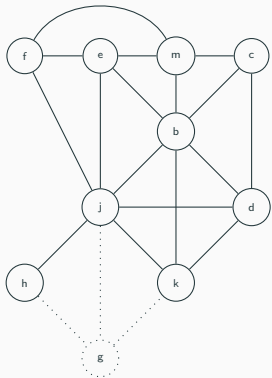
Given K available colors, a graph G and one node N . If the degree of N is less than K then if we can color $G \setminus N$ then we can color G

Let $K=3$.



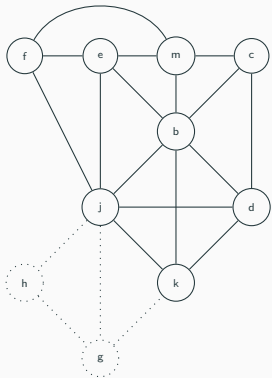
Simplification of the interference graph

g



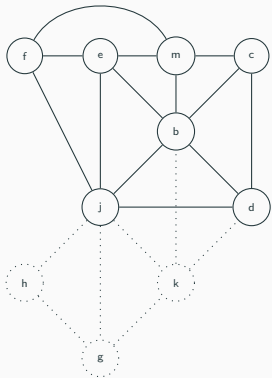
Simplification of the interference graph

g h



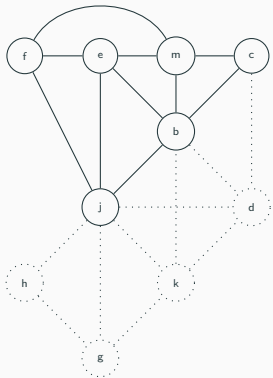
Simplification of the interference graph

g h k



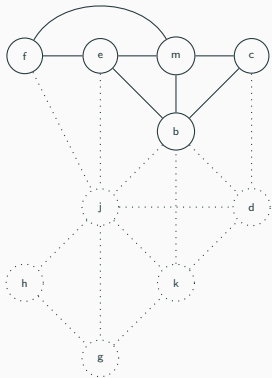
Simplification of the interference graph

g h k d



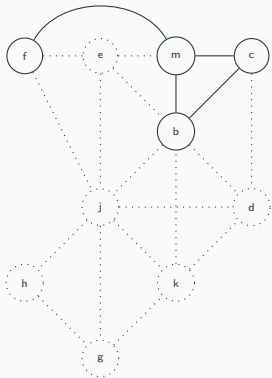
Simplification of the interference graph

g h k d j



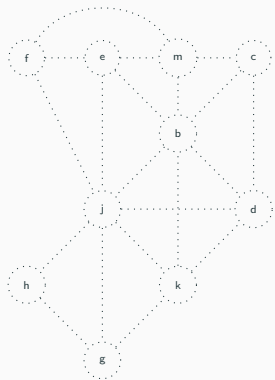
Simplification of the interference graph

g h k d j e



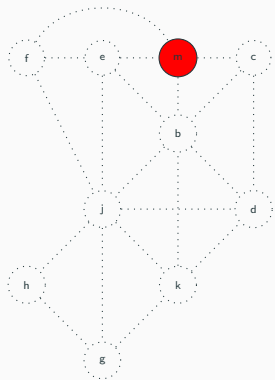
Simplification of the interference graph

g h k d j e f b c m



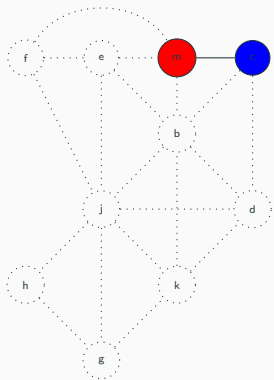
Coloring

g h k d j e f b c m



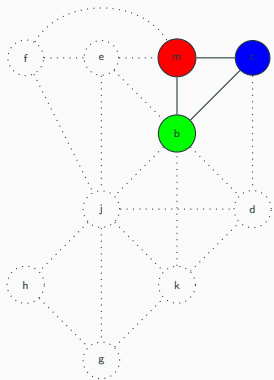
Coloring

g h k d j e f b c



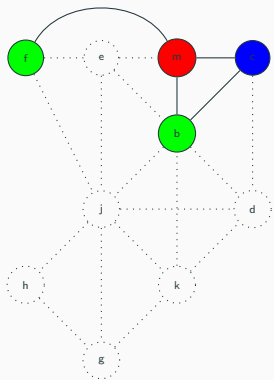
Coloring

g h k d j e f b

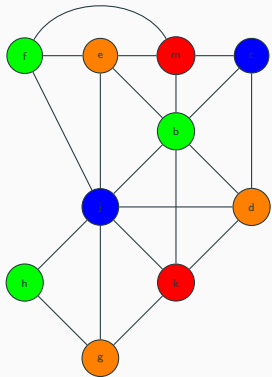


Coloring

g h k d j e f



Coloring



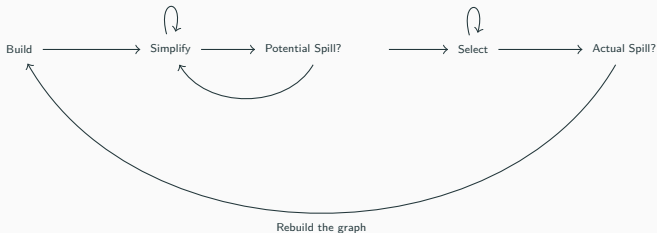
Simple allocation

→ Build → Simplify → Select

- Build : We build the interference graph
- Simplify: We remove one by one low degree ($< K$) nodes
- Select: We color the graphs by rebuilding back the graph
 - A color not used by node's neighbors is chosen

- The above heuristic may not work:
 - During simplify phase, all nodes are of degree $\geq K$.
- Solution: spill some value to memory
 - Allocate one cell on the stack
 - Each time the value is accessed, we read from and store it back to the stack
 - Reduces the lifetime of the value and therefore reduces its degree on G

Simplification with Spilling



- Opportunistic: Not all potential spills translate to actual spills during coloring phase
- Choosing which register to spill should be done with care: eg. do not spill a loop iteration variable.