Register Allocation

Ding Yaoyao
yyding@sjtu.edu.cn

Register Allocation

- Why？
- Why important？
- How？

Naive Register Allocation

- All virtual registers store in memory
- Fetch into physical register when needed and store back after operation

Graph Register Allocation

- Liveness Analysis

- Approximate algorithm

Liveness Analysis

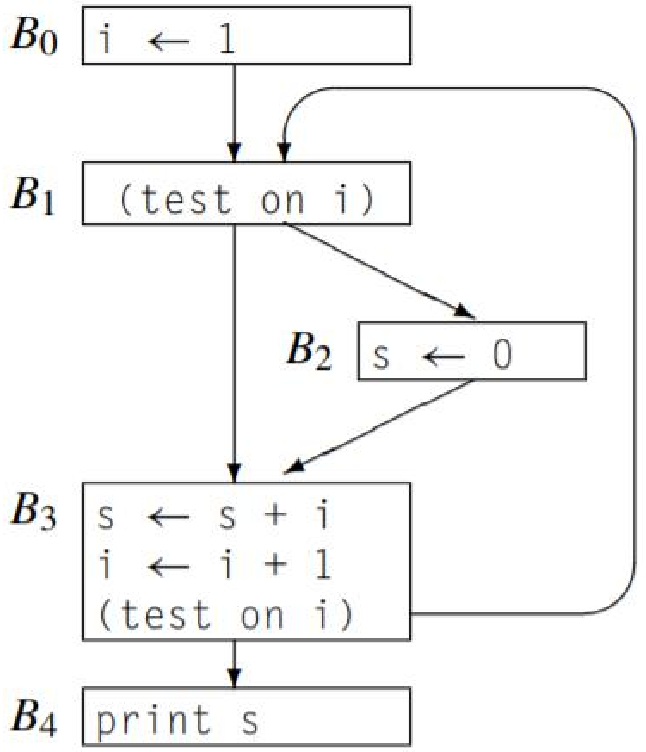
- A variable v is **live** at point p if and only if there exists a path in the CFG from p to a use of v along which v is not redefined.

- **UEVar**(m) contains the upward–exposed variables in m (those variables that are used in m before any redefinition in m)

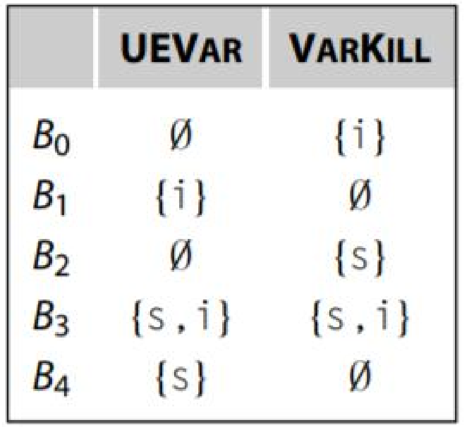
- **VarKill**(m) contains all the variables that are defined in m

- For each block b, define **LiveOut**(b) as all the variables that are live on exit from b

Example

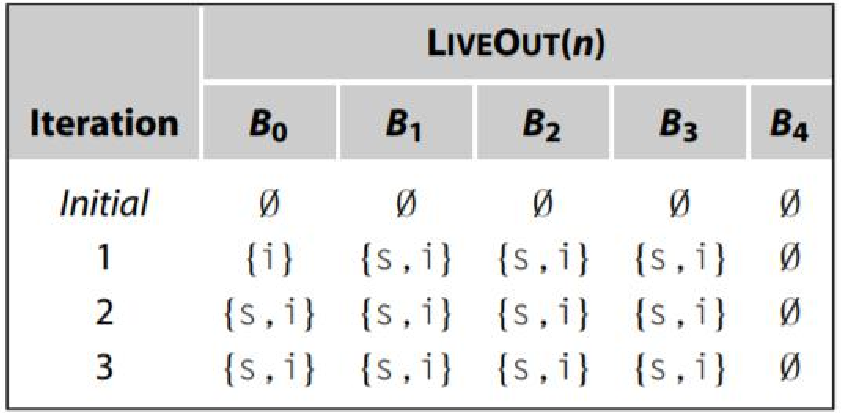


(a) Example Control-Flow Graph



	UEVAR	VARKILL
B_0	∅	{i}
B_1	{i}	∅
B_2	∅	{s}
B_3	{s,i}	{s,i}
B_4	{s}	∅

(b) Initial Information



Iteration	LIVEOUT(*n*)				
	B_0	B_1	B_2	B_3	B_4
Initial	∅	∅	∅	∅	∅
1	{i}	{s,i}	{s,i}	{s,i}	∅
2	{s,i}	{s,i}	{s,i}	{s,i}	∅
3	{s,i}	{s,i}	{s,i}	{s,i}	∅

(c) Progress of the Solution

Algorithm

$$\text{LIVEOUT}(n) = \bigcup_{m \in succ(n)} (\text{UEVAR}(m) \cup (\text{LIVEOUT}(m) \cap \overline{\text{VARKILL}(m)}))$$

```
// assume block b has k operations
// of form ‘‘x ← y op z’’
for each block b
    Init(b)

Init(b)
    UEVar(b) ← ∅
    VarKill(b) ← ∅
    for i ← 1 to k
        if y ∉ VarKill(b)
            then add y to UEVar(b)
        if z ∉ VarKill(b)
            then add z to UEVar(b)
        add x to VarKill(b)
```

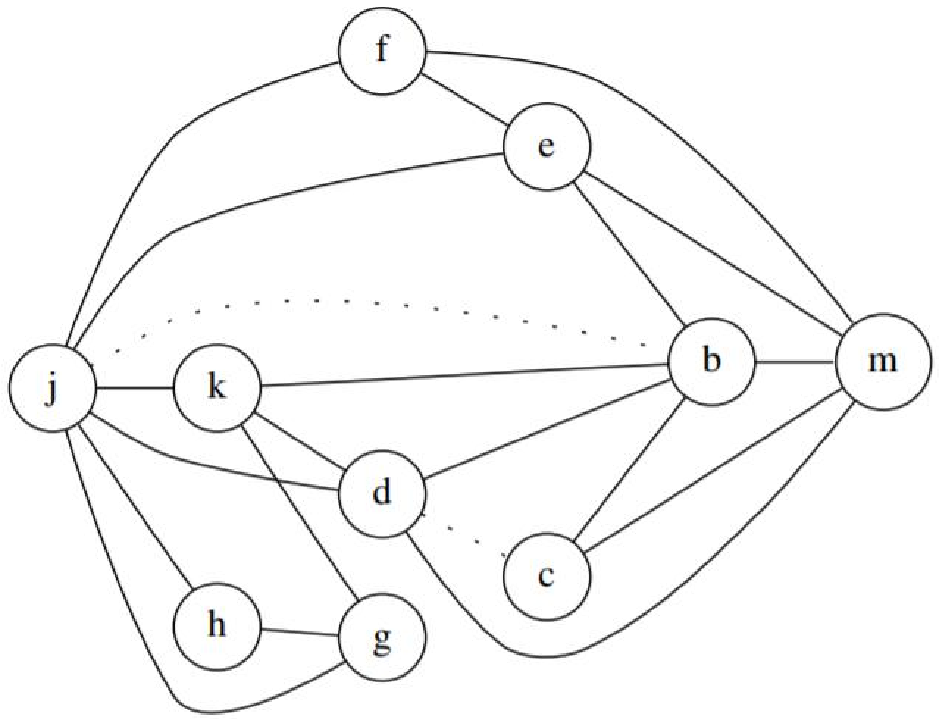
(a) Gathering Initial Information

```
// assume CFG has N blocks
// numbered 0 to N-1
for i ← 0 to N-1
    LiveOut( i ) ← ∅
changed ← true
while (changed)
    changed ← false
    for i ← 0 to N-1
        recompute LiveOut( i )
        if LiveOut( i ) changed then
            changed ← true
```

(b) Solving the Equations

Interference Graph

```
live-in:  k j
        g := mem[j+12]
        h := k - 1
        f := g * h
        e := mem[j+8]
        m := mem[j+16]
        b := mem[f]
        c := e + 8
        d := c
        k := m + 4
        j := b
live-out:  d k j
```



Simple Graph Register Allocation

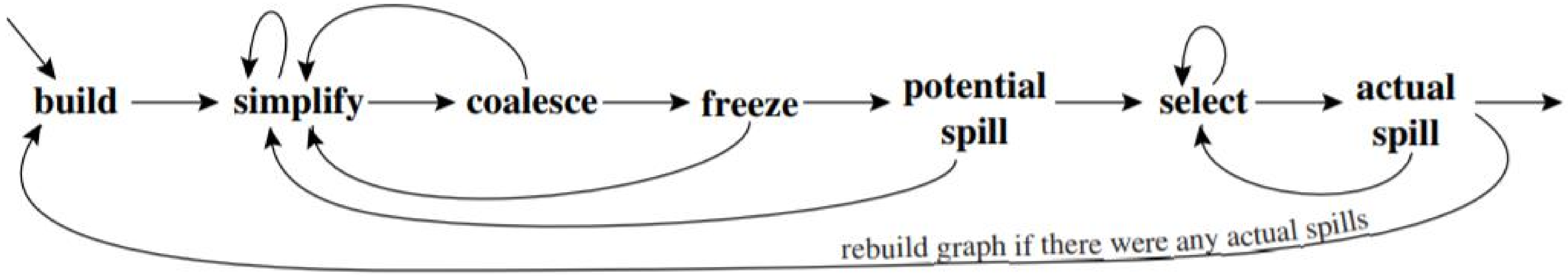
- Build – construct interference graph
- Simplify – reduce to a smaller problem
- Spill – (potential spill)
- Select – assign colors (physical register)

Simple Graph Register Allocation

- While True:
  - Build Interference Graph
  - Simplifylist = Nodes Whose Degree < K
  - Spilllist = Nodes Whose Degree >= K
  - While Simplifylist Or Spilllist Not Empty:
    - If Simplifylist Not Empty: Simplify()
    - Else: Potentialspill()
  - Assigncolor()
  - If Success: Break
  - Else: Realspill()

More Advanced Graph Register Allocation

- A piece of cake for you.



Reference

- Liveness Analysis: Engineering A Compiler
- Graph Register Allocation: Tiger Book
- Zhihu: https://www.zhihu.com/question/29355187