
Análise de Longevidade

Sandro Rigo
sandro@ic.unicamp.br

Introdução

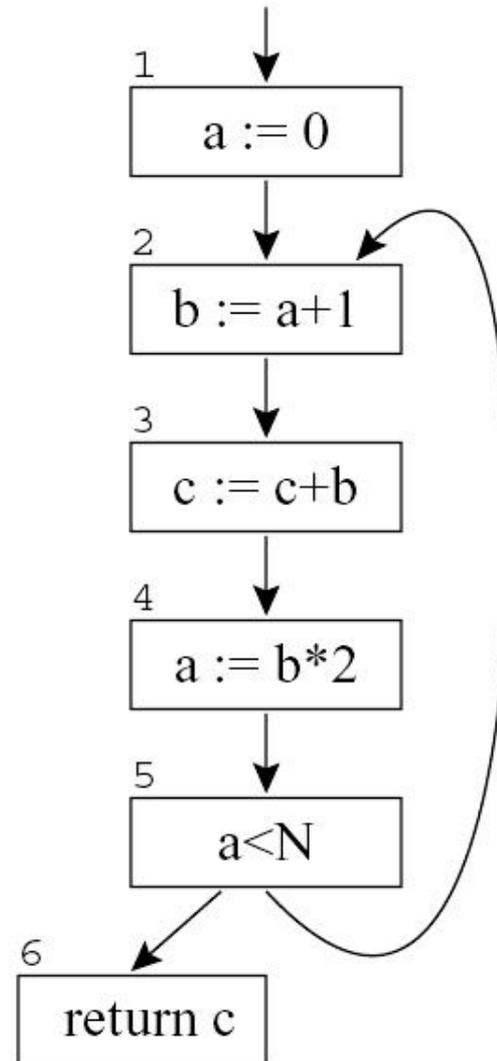
- Linguagem intermediária
 - Gerada pelo front-end considerando número infinito de registradores para temporários
- Máquinas reais têm finitos registradores
 - Para máquinas RISC, 32 é um número típico
- Dois valores temporários podem ocupar o mesmo registrador se não estão “em uso” ao mesmo tempo
 - Muitos temporários podem caber em poucos registradores
 - Os que não couberem vão para a memória (spill)

Introdução

- O compilador analisa a IR para saber quais valores estão em uso ao mesmo tempo
- Chamamos de *viva* uma variável que pode vir a ser usada no futuro
- Esta tarefa então, é conhecida como *liveness analysis*

Control Flow Graph (CFG)

```
a ← 0
L1 : b ← a + 1
      c ← c + b
      a ← b * 2
      if a < N goto L1
      return c
```

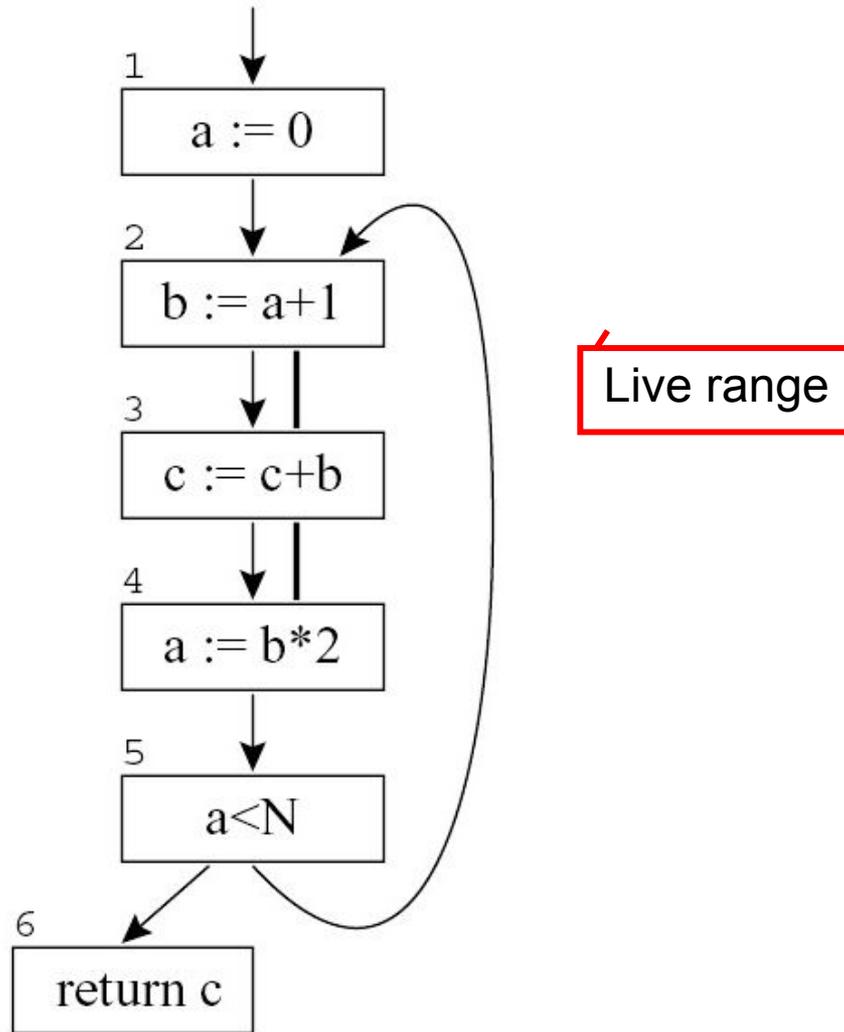


Análise de Longevidade

- b é usada em 4
 - Precisa estar viva na aresta $3 \rightarrow 4$
- b não é definida (atribuída) no nó 3
 - Logo, deve estar viva na aresta $2 \rightarrow 3$
- b é definida em 2
 - Logo, b está morta na aresta $1 \rightarrow 2$
 - Seu valor nesse ponto não será mais útil a ninguém
- Live range de b :
 - $\{2 \rightarrow 3, 3 \rightarrow 4\}$

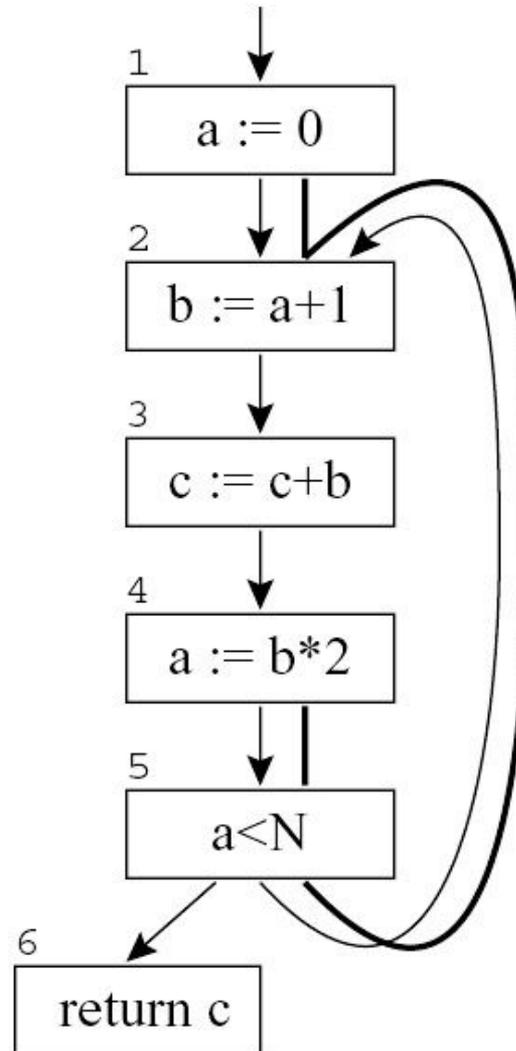
Análise de Longevidade

Como seria para a e c?



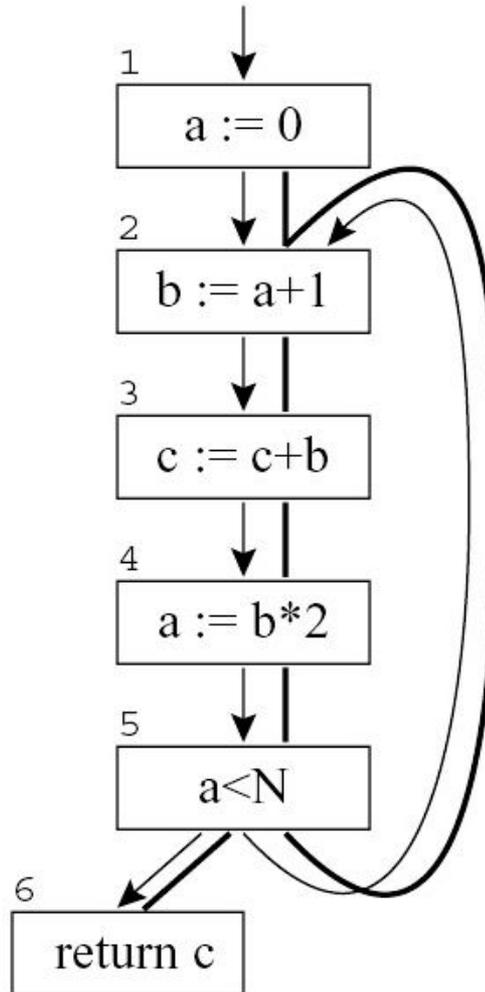
Análise de Longevidade

A: {1 → 2, 4 → 5,
4 → 5, 5 → 2}



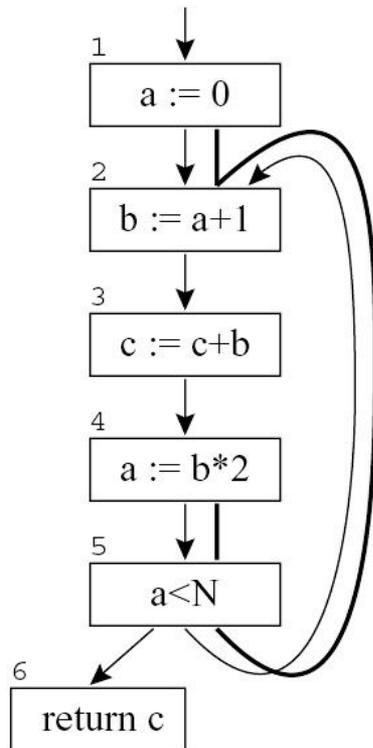
Análise de Longevidade

Alguma coisa especial sobre c ?

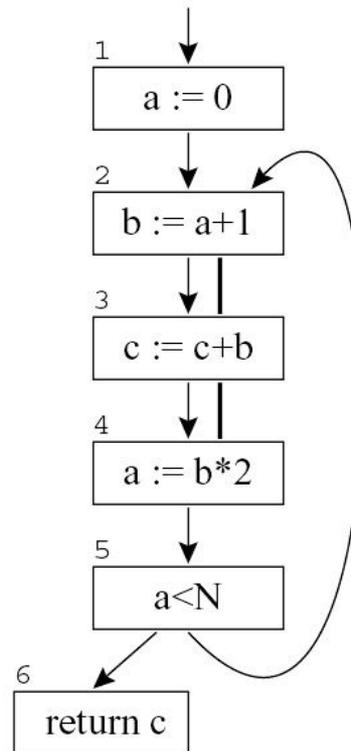


Análise de Longevidade

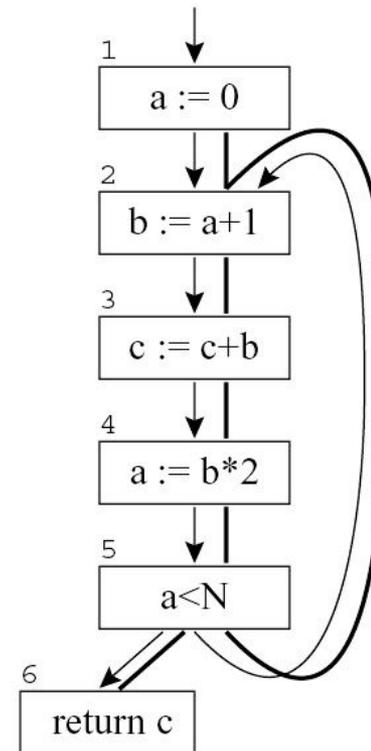
- De quantos registradores preciso?



(a)



(b)



(c)

Análise de Longevidade

- É um exemplo de análise de fluxo de dados
 - Dataflow Analysis
- Terminologia:
 - Succ[n]: conjunto de nós sucessores a n
 - Pred[n]: conjunto de predecessores de n
 - Out-edges: saem para os sucessores
 - In-edges: chegam dos predecessores
 - Uma atribuição a uma variável define a mesma
 - Uma ocorrência do lado direito de uma expressão é um uso da variável

Análise de Longevidade

- Terminologia:

- Def de uma variável é o conjunto de nós do grafo que a definem
- Def de um nó é o conjunto de variáveis que ele define
- Analogamente para use

- Longevidade:

- Uma variável v está viva em uma aresta se existe um caminho direcionado desta aresta até um uso de v , que não passa por alguma definição de v
- Live-in: v é live-in em um nó n se v está viva em alguma in-edge de n
- Live-out: v é live-out em n se v está viva em alguma out-edge de n

Computando Liveness

1. Se v está em $use[n]$, então v é *live-in* em n .
2. Se v é *live-in* no nó n , então ela é *live-out* para todo m em $pred[n]$.
3. Se v é *live-out* no nó n , e não está em $def[n]$, então v é também *live-in* em n .

Algoritmo

$$in[n] = use[n] \cup (out[n] - def[n])$$
$$out[n] = \bigcup_{s \in succ[n]} in[s]$$

for each n

 in[n] {}; out[n] {}

repeat

 for each n

 in'[n] \leftarrow in[n]; out'[n] \leftarrow out[n]

 in[n] \leftarrow use[n] \cup (out[n] - def[n])

 out[n] \leftarrow $\bigcup_{s \in succ[n]} in[s]$

until in'[n] = in[n] and out'[n] = out[n]

 for all n

Algoritmo

- Execute o algoritmo para o grafo do exemplo anterior
- Temos como melhorar o desempenho?
- Sim:
 - Usando uma ordem melhor para os nós
 - Repare que $in[i]$ é calculado a partir de $out[i]$ e $out[i-1]$ é computado a partir de $in[i]$
 - A convergência ocorre antes de computarmos
 - $Out[i], in[i], out[i-1], \dots$
 - Invertendo a ordem dos nós aproveitamos mais cedo as informações!

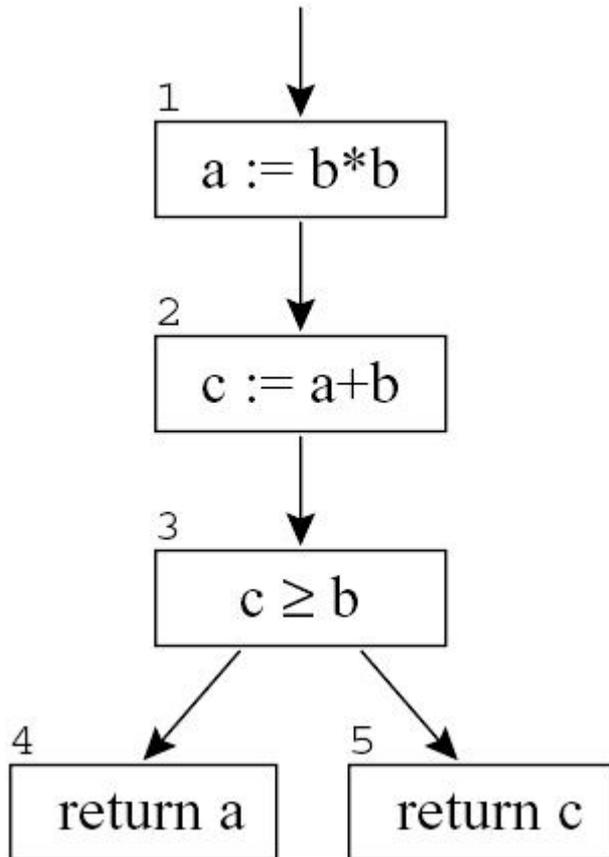
Algoritmo

- O fluxo da análise deve seguir o fluxo do liveness: backwards
- A ordenação pode ser obtida através de uma busca em profundidade
- Complexidade:
 - Pior caso: $O(N^4)$
 - Com a ordenação, na prática roda tipicamente entre $O(N)$ e $O(N^2)$

Algoritmo

- **É conservativo:**
 - Se uma variável pode estar viva em algum nó n , ela estará no $\text{out}[n]$
 - Pode haver alguma variável em $\text{out}[n]$ que na verdade não seja realmente usada adiante
- Deve ser dessa maneira para prevenir o compilador de tornar o programa errado!

Exemplo



- Qual seria o conjunto $in[4]$?
- E o $out[3]$?
- Algo estranho?

Blocos Básicos

- Nós com apenas um predecessor e um sucessor podem ser unidos
- Teremos um grafo com menos nós
- Cada nó é um bloco básico
- Os algoritmos de dataflow funcionam mais rapidamente
-

Grafo de Interferência

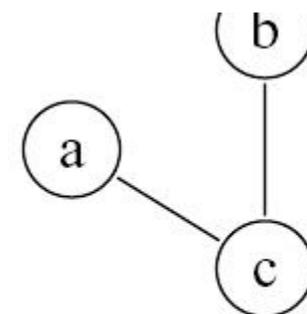
- A informação de liveness é usada para otimização
 - Alocação de registradores
- Interferência: ocorre quando a e b não podem ocupar o mesmo registrador
 - Live ranges com sobreposição
 - a não pode ser alocada a r1

Grafo de Interferência

- Representação:

	a	b	c
a			x
b			x
c	x	x	

(a) Matrix



(b) Graph

Grafo de Interferência

- MOVE: é importante não criar falsas interferências entre a fonte e destino

$$\begin{array}{ll} t \leftarrow s & (\text{copy}) \\ \vdots & \\ x \leftarrow \dots s \dots & (\text{use of } s) \\ \vdots & \\ y \leftarrow \dots t \dots & (\text{use of } t) \end{array}$$

- S e t estariam vivas após a instrução de cópia
- Devemos aproveitar o mesmo registrador

Grafo de Interferência

1. Definição de a que não seja move:

1. Live-out = b_1, \dots, b_j
 1. Adicione as arestas $(a, b_1), \dots, (a, b_j)$.

2. Moves $a \leftarrow c$:

1. Live-out = b_1, \dots, b_j
 1. Adicione as arestas $(a, b_1), \dots, (a, b_j)$ para os b_i 's que não são o mesmo que c