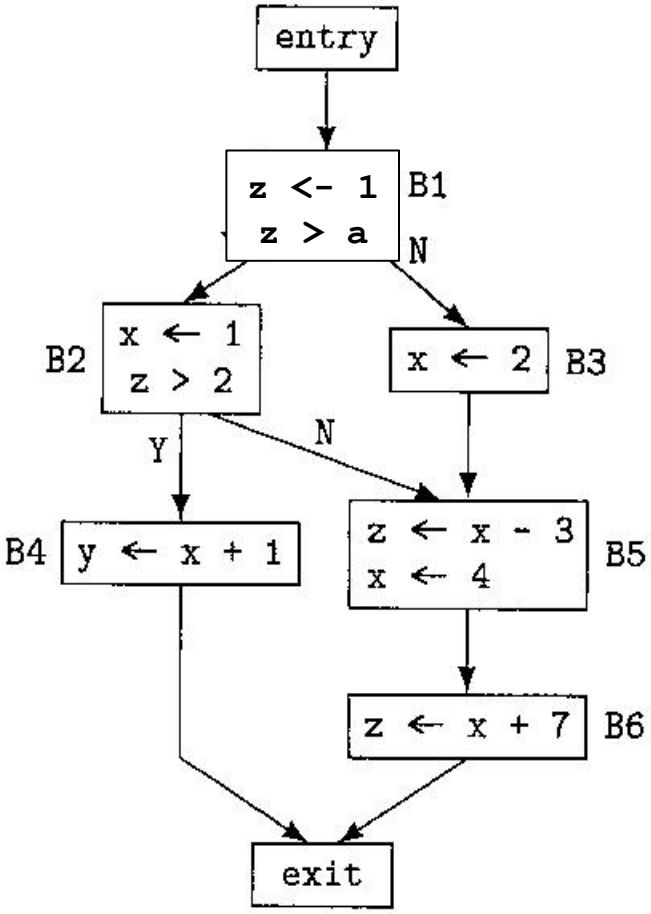
Static Single Assignment Form (SSA)

Sandro Rigo
sandro@ic.unicamp.br

Introdução

- Uma forma de IR
- Separa os valores operados das localidades onde são armazenados
- Codifica tanto o fluxo de dados como o de controle diretamente na IR
- Torna possível versões mais eficientes de algumas otimizações

Introdução



- Ideia:
  - Garantir que, ao longo de cada aresta do CFG, o valor corrente da variável x esteja associado a um único nome

Introdução

- Na forma SSA
  - Cada definição no procedimento criará um único nome
  - Cada uso será referente a uma única definição
  - Du-chains estão explícitas no CFG
    - Usos e definições possuem exatamente o mesmo no para a variável
- Otimizações beneficiadas
  - Constant Propagation
  - Code Motion
  - Strength Reduction
  - Etc;

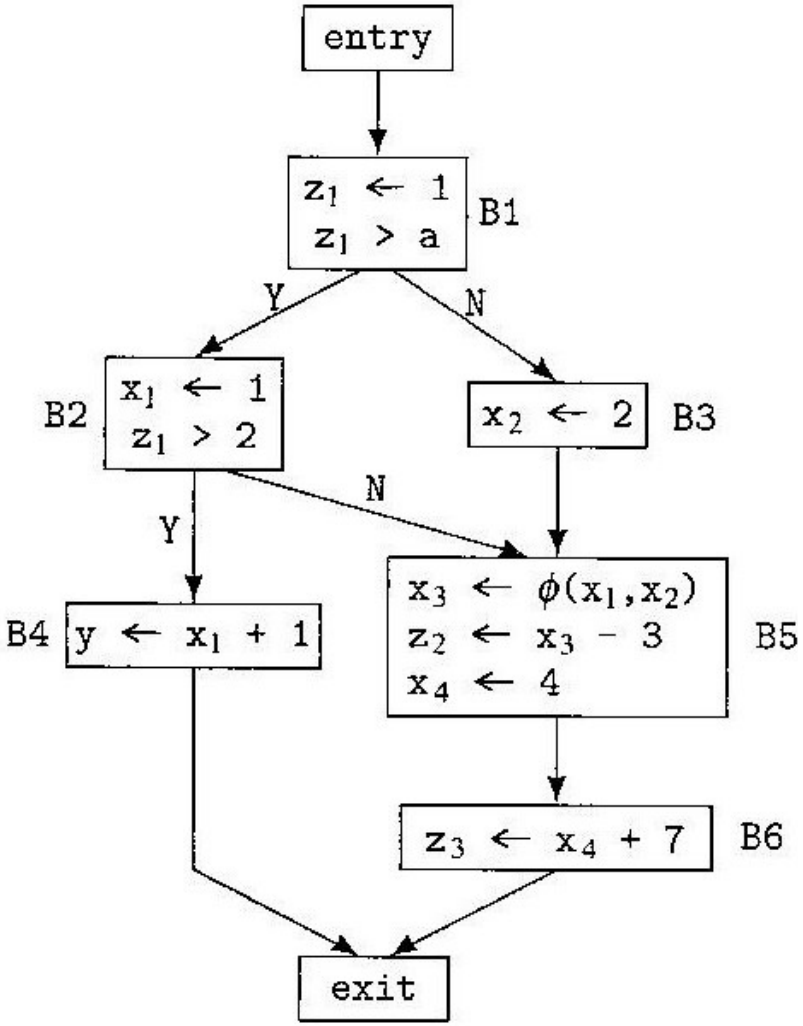
Introdução

- Método
  - Incluir novas atribuições
    - Novos nomes a x, criando índices
    - Somente nos pontos onde existem dúvidas
      - Como escolher o valor a ser atribuído?
        - » Φ-functions

Φ-functions

- Argumentos
  - Todos os nomes para os valores associados a cada aresta que entra no bloco
  - Escritos da esquerda para à direita, correspondendo as arestas da esquerda para a direita
- Avaliação
  - Quando o controle passa ao bloco
    - Todas são avaliadas concorrentemente
    - Recebem o valor correspondente à aresta pelo qual o controle entrou no bloco

Exemplo


entry
$z_1 \leftarrow 1$
$z_1 > a$
B1
Y
N
B2
$x_1 \leftarrow 1$
$z_1 > 2$
$x_2 \leftarrow 2$
B3
N
Y
B4
$y \leftarrow x_1 + 1$
$x_3 \leftarrow \phi(x_1, x_2)$
$z_2 \leftarrow x_3 - 3$
$x_4 \leftarrow 4$
B5
$z_3 \leftarrow x_4 + 7$
B6
exit

Tradução para SSA

- Identifique os joint points
  - Pontos do CFG onde múltiplos caminhos convergem
    - Local para inserção de Φ-functions
- Inserir Φ-functions triviais nos joint points
    - Φ (x,x,...,x): no. de argumentos igual ao número de predecessores pelos quais chega uma definição de x
    - Isto para cada nome x que o código define ou usa no procedimento
    - Pode inserir em qualquer ordem
    - Todas as Φ-functions do inicio de um bloco executam concorrentemente

Tradução para SSA

- Renomear
  - Renomear usos e definições de x para estabelecer a propriedade SSA
  - Usa reaching definitions
- Esse método insere muitas Φ-functions
  - Φ (xi,xi)
  - Φ (x,x,...,x) que não estão vivas
  - Chamado de SSA maximal

SSA Mínima

- Gerar SSA com o mínimo de Φ-functions
- Compreender quais variáveis precisam da Φ-function em cada joint point
  - Determinar, para cada definição, o conjunto de joint-points que precisam de uma Φ-function para o valor criado por esta definição

Fronteira de Dominância

- Idéia:
  - Seja uma definição em um nó n do CFG. Este valor potencialmente atinge:
    - Todo *m* onde *n* ϵ Dom(*m*)
    - Não precisa de Φ-function
  - Só não acontece se existir algum nó *p* entre *n* e *m* com outra definição para o mesmo nome.
    - Neste caso, *p* está forçando a presença de Φ-function, não *n*
  - Uma definição em um nó *n* força uma Φ-function em joint-nodes que residem imediatamente fora da região do CFG que *n* domina

Fronteira de Dominância

- Uma definição em *n* força uma Φ-function em qualquer join node *m* onde
  - *n* domina um predecessor de *m*
    - (*q* ϵ Preds(*m*) e *n* ϵ Dom(*q*))
  - *n* não domina estritamente *m*, isto é, *n* !sdom *m*
    - *n* sdom *m* => *n* dom *m* e *n* ≠ *m*
    - Essa noção permite uma Φ-function no início de um laço de um único bloco
- Conjunto de nós *m* respeitando essa definição é a DF(n)

Fronteira de Dominância

- DF(n):
  - Conjunto dos 1ºs nós alcaçáveis a partir de n, que n não domina, em cada caminho do CFG a partir de n
- Observações
  - Nós de uma DF devem ser join-nodes no CFG
  - Os predecessores de qualquer join-node j devem ter j em sua DF a menos que o predecessor domine j
  - Os dominadores dos predecessores de j devem ter j em suas DF's, a menos que também dominem j

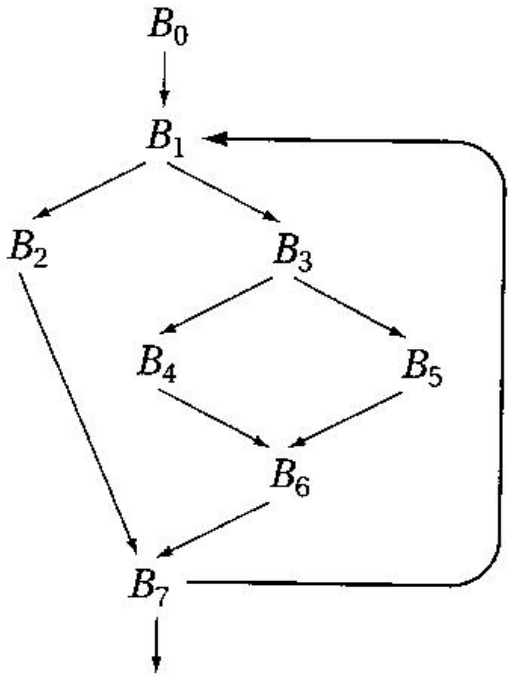
$$DF(x) = \{y \mid (\exists z \in Pred(y) \text{ tal que } x \; dom \; z) \text{ e } x \; !sdom \; y\}$$

Algoritmo (Cooper & Torczon)

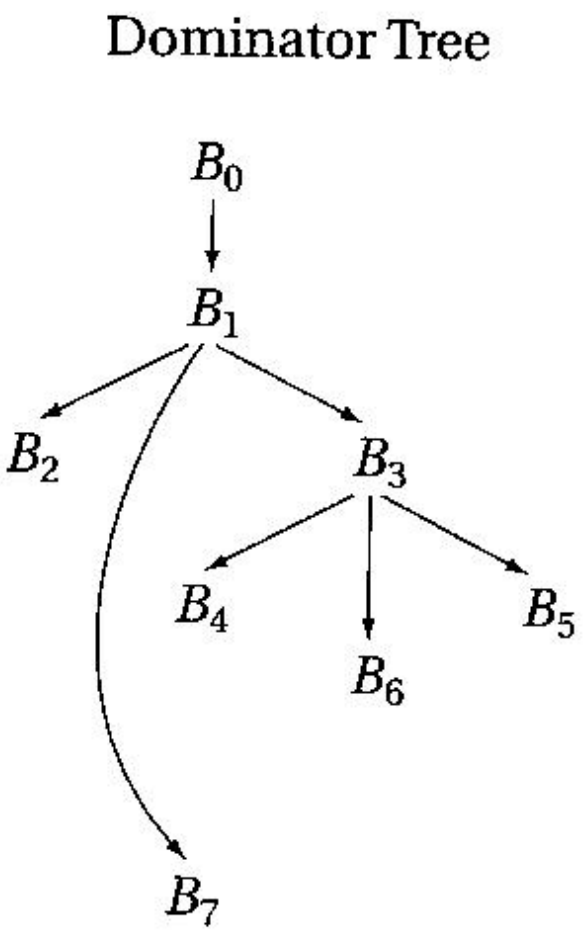
- For all nodes, n
  - DF(n) = {}
- For all nodes, n
  - If n has multiple predecessors
    - For each predecessor p of n
      - runner = p
      - While runner != IDom(n)
        - » DF (runner) = DF (runner) U {n}
        - » runner = IDom(runner)

Exemplo

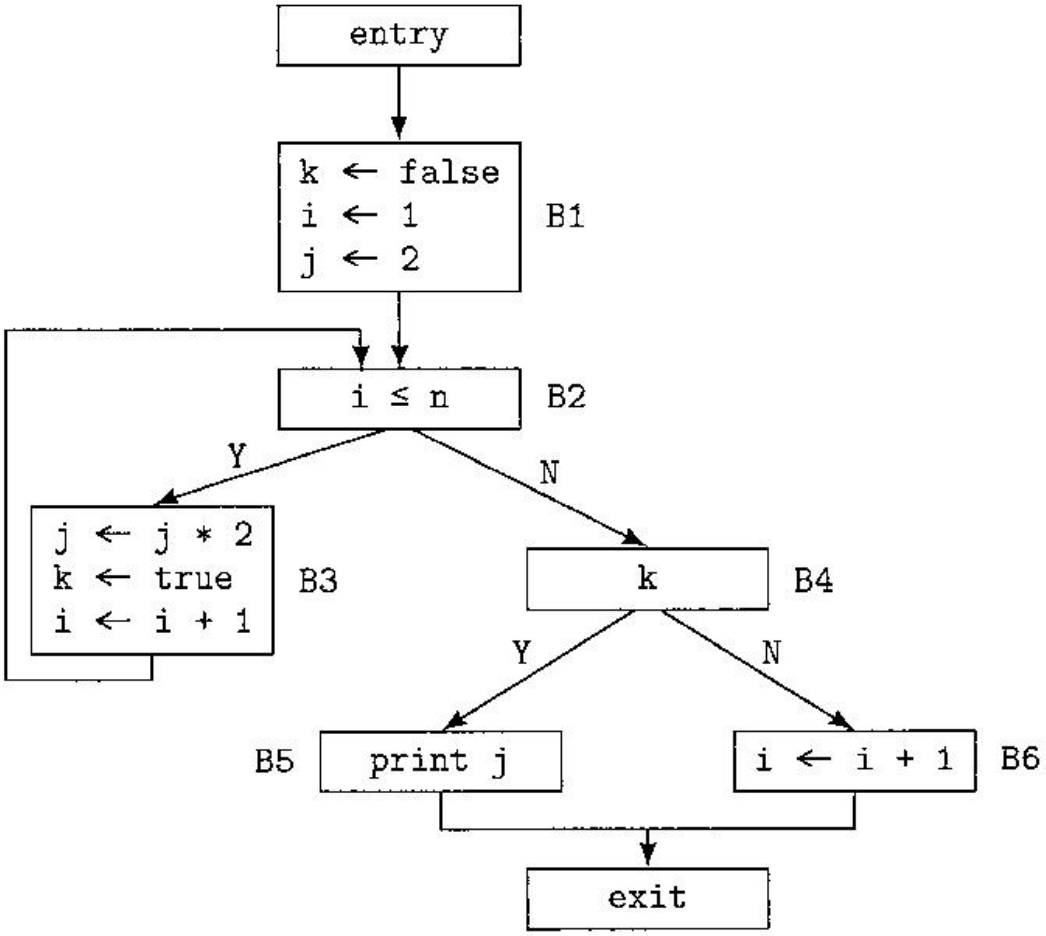
Control-Flow Graph



Exemplo



Exemplo


entry
k ← false
i ← 1
j ← 2
B1
i ≤ n
B2
Y
N
j ← j * 2
k ← true
i ← i + 1
B3
k
B4
Y
N
B5
print j
i ← i + 1
B6
exit

Exemplo

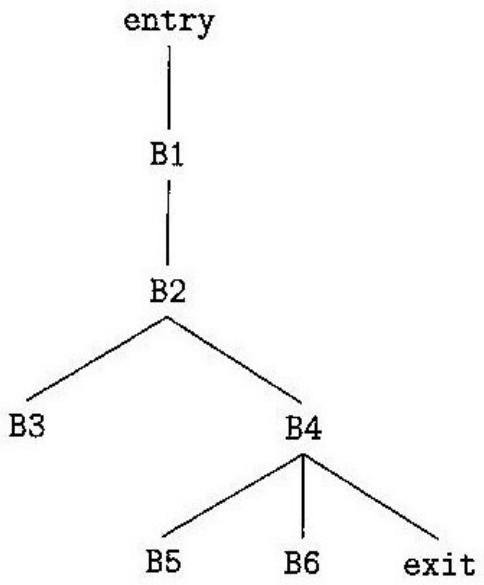
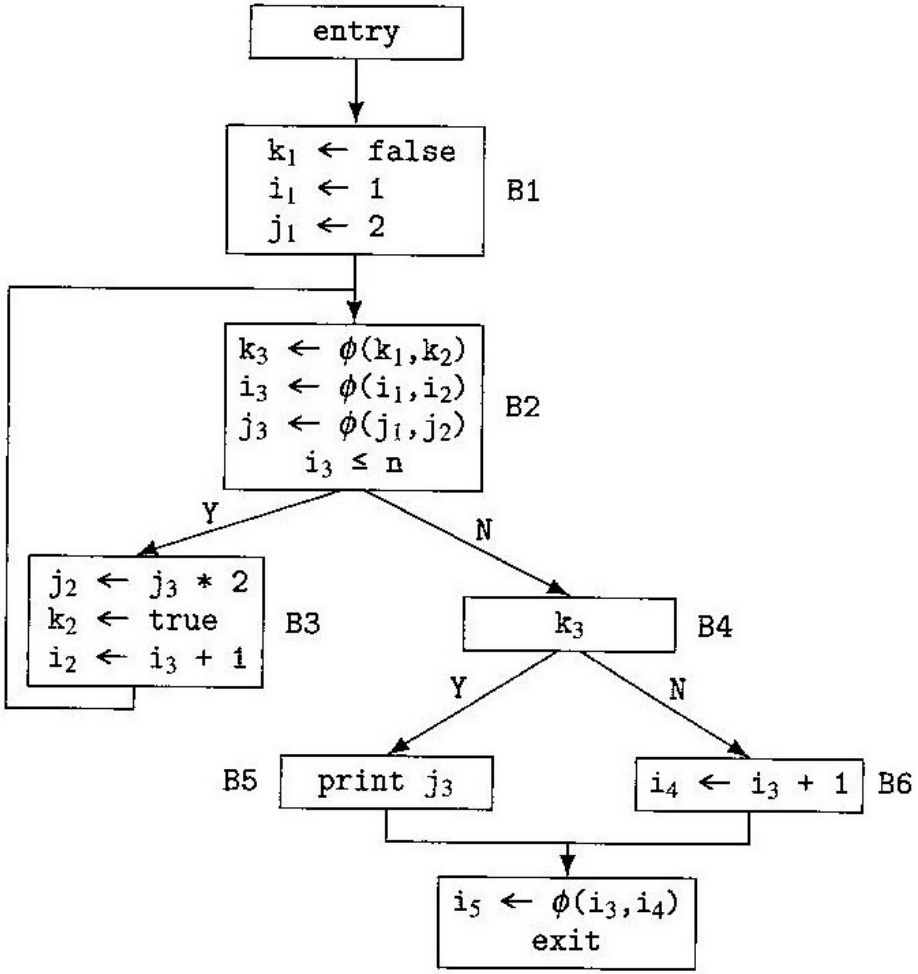


FIG. 8.22 Dominator tree for the flowgraph in Figure 8.21.

Exemplo


entry
$k_1 \leftarrow$ false
$i_1 \leftarrow 1$
$j_1 \leftarrow 2$
B1
$k_3 \leftarrow \phi(k_1,k_2)$
$i_3 \leftarrow \phi(i_1,i_2)$
$j_3 \leftarrow \phi(j_1,j_2)$
$i_3 \leq n$
B2
Y
N
$j_2 \leftarrow j_3 * 2$
$k_2 \leftarrow$ true
$i_2 \leftarrow i_3 + 1$
B3
k_3
B4
Y
N
B5
print j_3
$i_4 \leftarrow i_3 + 1$
B6
$i_5 \leftarrow \phi(i_3,i_4)$
exit