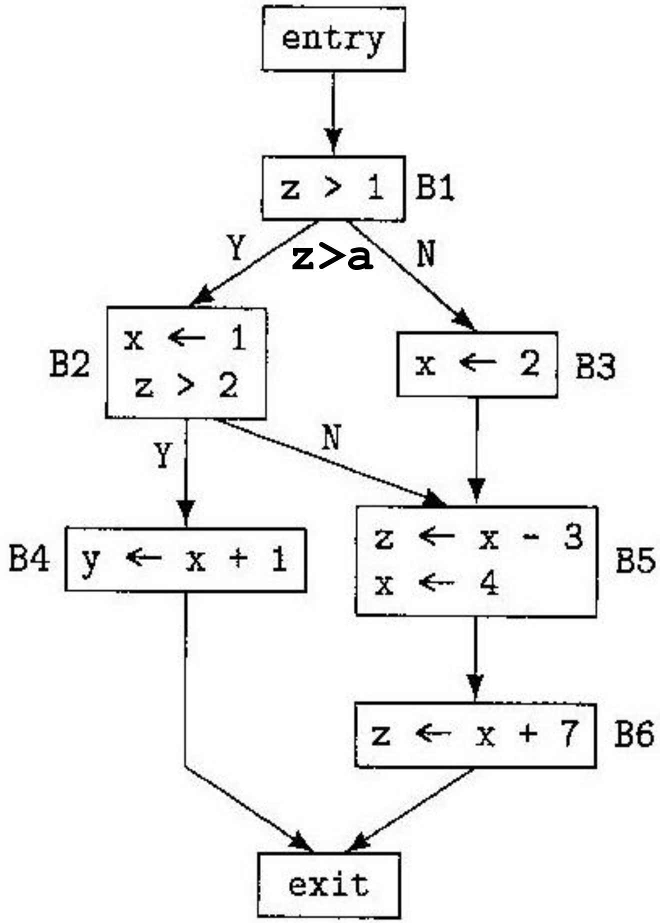
Static Single Assignment Form (SSA)

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



- Idéia:
  - 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;

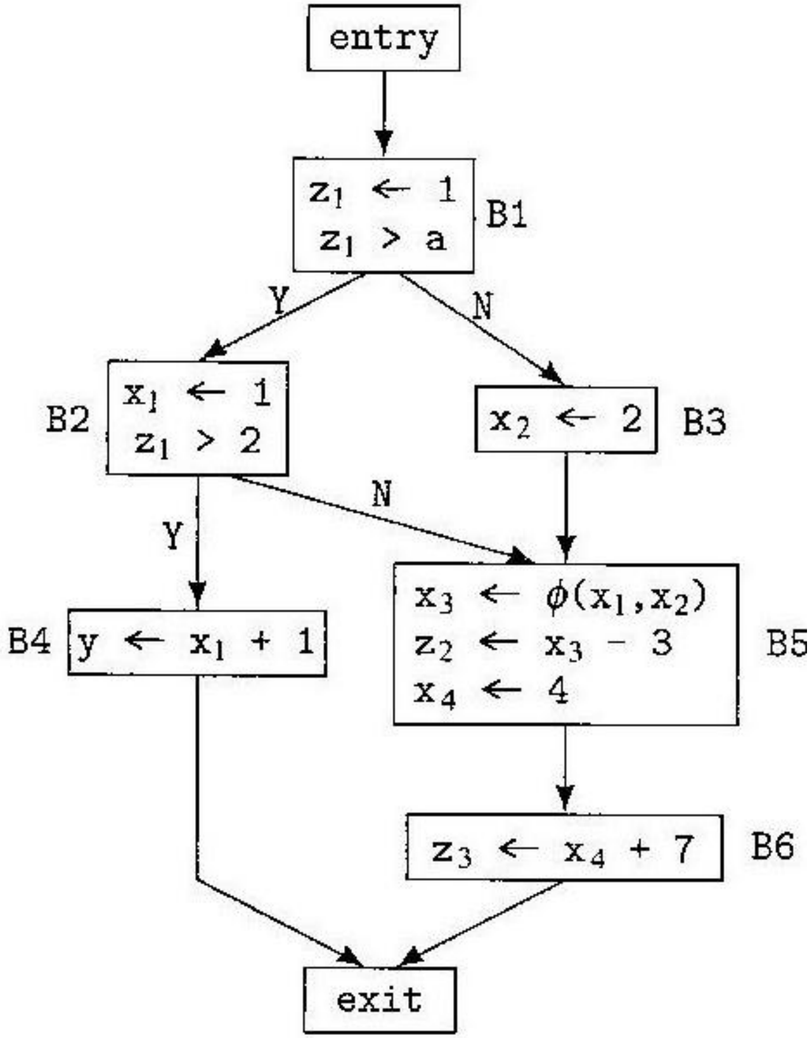
Introdução

- 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

Tradução para SSA

- A ordem das Φ-functions não é importante
  - Lembre-se que todas são avaliadas concorrentemente
- Renomear
  - Computar reaching definitions
  - Renomear usos e definições de x para estabelecer a propriedade SSA
- Este método insere Φ-functions desnecessárias
  - Diminuem a precisão de algumas análises
  - Ocupam memória

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* є Dom(*m*)
    - Não precisa de Φ-function
  - Só não acontece de 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 $n \neq m$
    - 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)

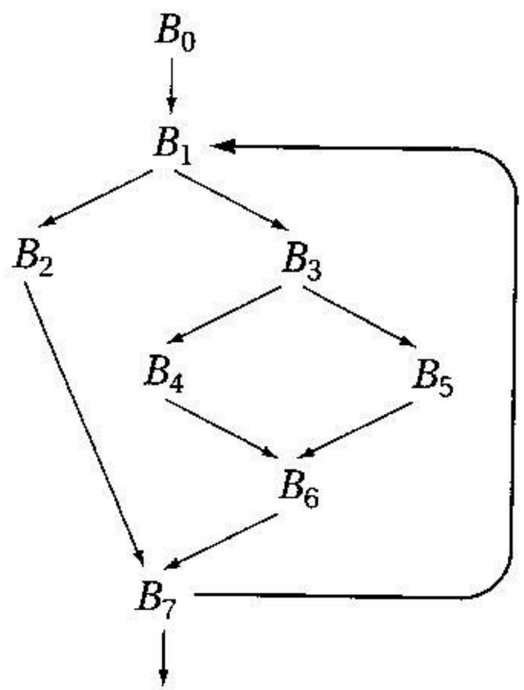
Fronteira de Dominância

- 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

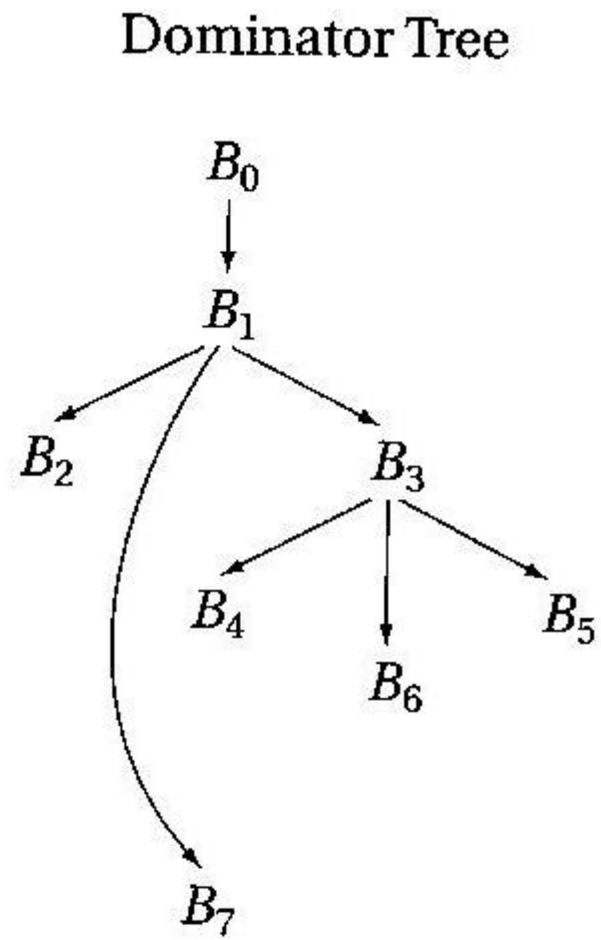
$$DF(x) = \{y \mid (\exists z \in Pred(y) \text{ tal que } x \; dom \; z) \text{ e } x \,!sdom\; y\}$$

Exemplo

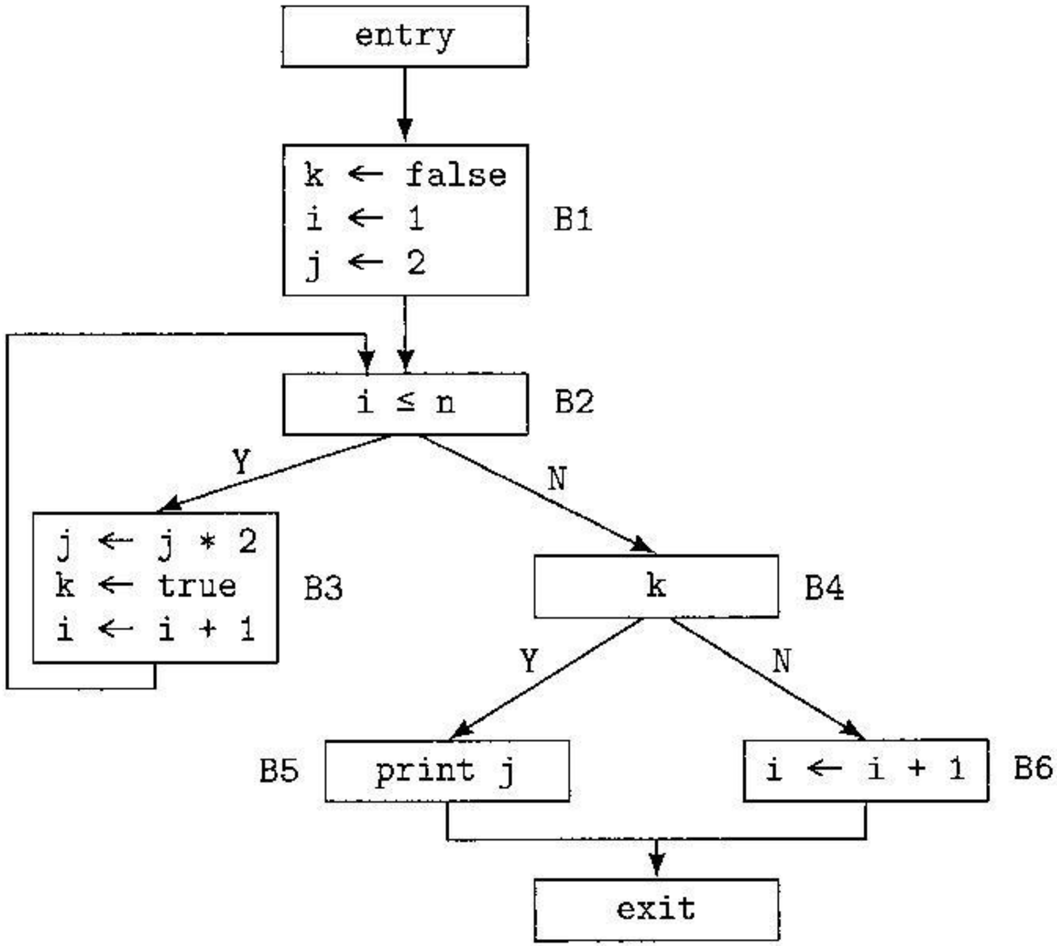
Control-Flow Graph



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

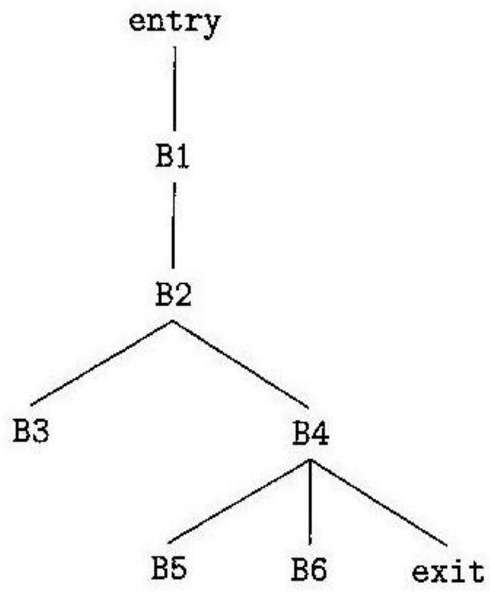
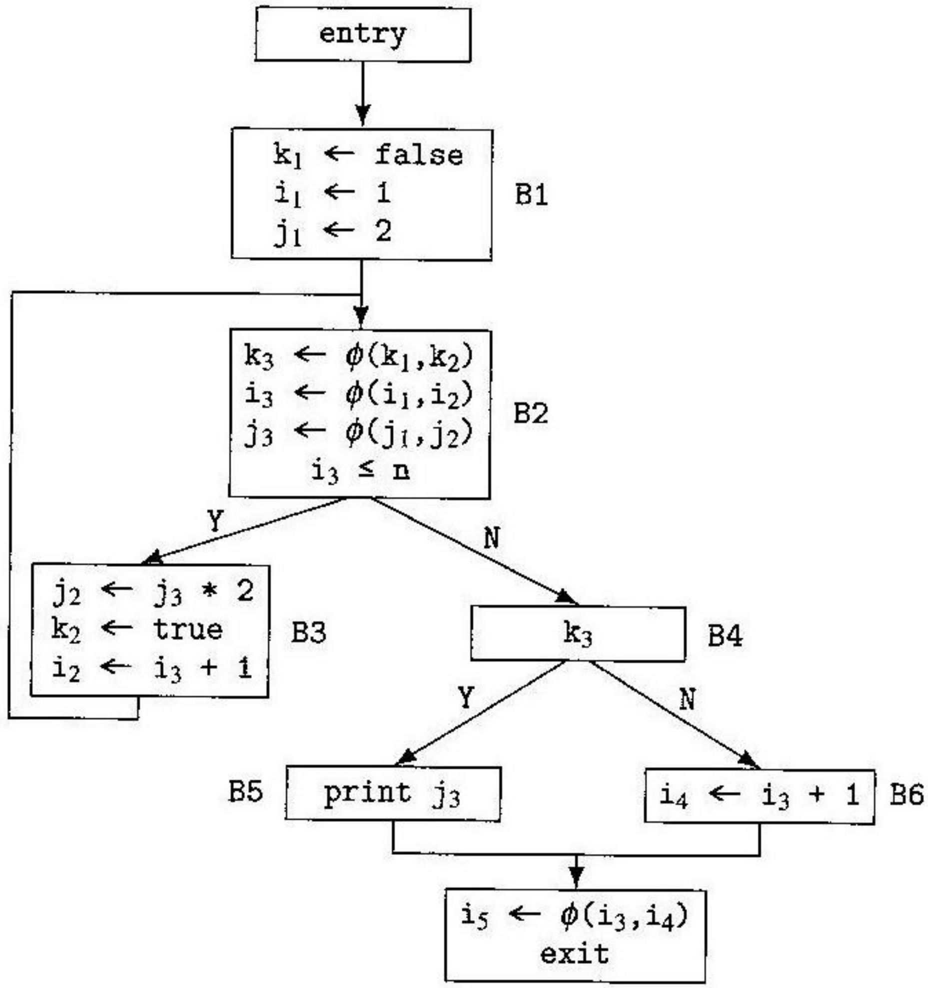


FIG. 8.22 Dominator tree for the flowgraph in Figure 8.21.

Exemplo


entry
k1 ← false
i1 ← 1
j1 ← 2
B1
k3 ← φ(k1,k2)
i3 ← φ(i1,i2)
j3 ← φ(j1,j2)
i3 ≤ n
B2
Y
N
j2 ← j3 * 2
k2 ← true
i2 ← i3 + 1
B3
k3
B4
Y
N
B5
print j3
i4 ← i3 + 1
B6
i5 ← φ(i3,i4)
exit

Fronteira de Dominância Iterada

- Para um conjunto de nós S temos:

$$DF(S) = \bigcup_{x \in S} DF(x)$$

- Fronteira de dominância iterada:

$$DF^{+}(S) = \lim_{i \to \infty} DF^{i}(S)$$

- Onde:

$$DF^{1}(S) = DF(S) \qquad DF^{i+1}(S) = DF(S \bigcup DF^{i}(S))$$

Fronteira de Dominância Iterada

- Seja S o conjunto de nós que atribui x, mais o nó de entrada temos:

$DF^{+}(S)$ é o conjunto de nós que precisa de Φ-functions para x

Exemplo

- Para k:
  - $DF^1(\{entry, B1,B3\}) = \{B2\}$
  - $DF^2(\{entry, B1,B3\}) = DF(\{entry, B1, B2, B3\}) = \{B2\}$
- Para i:
  - $DF^1(\{entry, B1,B3, B6\}) = \{B2,exit\}$
  - $DF^2(\{entry, B1,B3\}) = DF(\{entry, B1, B2, B3,B6,exit\}) = \{B2,exit\}$
- Para j:
  - $DF^1(\{entry, B1,B3\}) = \{B2\}$
  - $DF^2(\{entry, B1,B3\}) = DF(\{entry, B1, B2, B3\}) = \{B2\}$

Implementação

- DF(n):
  - Computar diretamente pela definição é quadrático no no. de nós do CFG
- Algoritmo linear
  - Quebra a computação em dois componentes
  - DF_{local} e DF_{up}

Fronteira de Dominância

- $DF_{local}(x)$:
  - Conjunto dos nós que seguem x imediatamente, mas x não os domina

$$DF_{local}(x) = \{y \in Succ(x) \,/\, idom(y) \neq x\}$$

- $DF_{up}(x,z)$:
  - Conjto de nós y que estão na DF de z, onde x é idom(z), mas x não é idom de y.

$$DF_{up}(x,z) = \{y \in DF(z) \,/\, idom(z) = x \wedge idom(y) \neq x\}$$

Fronteira de Dominância

$$DF_{local}(x) = \{y \in Succ(x) / idom(y) \neq x\}$$

$$DF_{up}(x,z) = \{y \in DF(z) / idom(z) = x \,\&\, idom(y) \neq x\}$$

$$DF(x) = DF_{local}(x) \bigcup \left[\bigcup_{z \in N(idom(z)=x)} DF_{up}(x,z) \right]$$

Fronteira de Dominância

```
IDom, Succ, Pred: Node ⟶ set of Node

procedure Dom_Front(N,E,r) returns Node ⟶ set of Node
    N: in set of Node
    E: in set of (Node × Node)
    r: in Node
begin
    y, z: Node
    P: sequence of Node
    i: integer
    DF: Node ⟶ set of Node
    Domin_Fast(N,r,IDom)
    P := Post_Order(N,IDom)
    for i := 1 to |P| do
        DF(P↓i) := ∅
        || compute local component
        for each y ∈ Succ(P↓i) do
            if y ∉ IDom(P↓i) then
                DF(P↓i) ∪= {y}
            fi
        od
        || add on up component
        for each z ∈ IDom(P↓i) do
            for each y ∈ DF(z) do
                if y ∉ IDom(P↓i) then
                    DF(P↓i) ∪= {y}
                fi
            od
        od
    od
    return DF
end    || Dom_Front
```

Fronteira de Dominância Iterada

```
procedure DF_Plus(S) returns set of Node
   S: in set of Node
begin
   D, DFP: set of Node
   change := true: boolean
   DFP := DF_Set(S)
   repeat
      change := false
      D := DF_Set(S ∪ DFP)
      if D ≠ DFP then
         DFP := D
         change := true
      fi
   until !change
   return DFP
end    || DF_Plus

procedure DF_Set(S) returns set of Node
   S: in set of Node
begin
   x: Node
   D := ∅: set of Node
   for each x ∈ S do
      D ∪= DF(x)
   od
   return D
end    || DF_Set
```