# Blocos Básicos e Grafos de Fluxo de Controle

**Sandro Rigo**
**sandro@ic.unicamp.br**

# Introdução

- Representação do código de 3 endereços em um **grafo** com o fluxo de controle é útil para entender os algoritmos de otimização.
- CFG: *control flow graph* – Grafo de Fluxo de Controle.

# Blocos Básicos

- Sequência maximal de instruções de 3-endereços consecutivas com as seguintes propriedades:
  - O fluxo de controle só pode entrar no bloco básico através da primeira instrução do bloco.
  - O fluxo de controle só pode deixar o bloco básico na última instrução do bloco.

```
t1 = a * a
t2 = a * b
t3 = b * 3
t4 = t2 - t3
```

# Algoritmo para gerar BBs

- Entrada: seqüência de código 3 endereços .
- Defina os líderes (iniciam os BBs):
  - Primeira instruçãodo código (e.g. função) é um líder
  - Todo alvo de um salto, condicional ou incondicional, é um líder
  - Toda instrução que sucede imediatamente um salto, condicional ou incondicional, é um líder
- Os BBs são compostos pelos líderes e todas as instruções subsequentes até o próximo líder (exclusive).

Dragão 8.4.1 – Basic Blocks

# BBs: Exemplo

```
1)   i=1
2)   j=1
3)   t1=10 * i
4)   t2 = t1 + j
5)   t3 = 8 * t2
6)   t4 = t3 – 88
7)   a[t4] = 0.0
8)   j = j + 1
9)   if j <= 10 goto (3)
10)  i = i + 1
11)  if i <= 10 goto (2)
12)  i = 1
13)  t5 = i – 1
14)  t6 = 88 * t5
15)  a[t6] = 1.0
16)  i = i + 1
17)  if i <= 10 goto (13)
```

; Matriz identidade

; double a[10,10]

```
for i from 1 to 10 do
    for j from 1 to 10 do
        a[i,j] = 0.0;
for i from 1 to 10 do
    a[i,i] = 1.0;
```

# BBs: Exemplo

```
1)   i=1
2)   j=1
3)   t1=10 * i
4)   t2 = t1 + j
5)   t3 = 8 * t2
6)   t4 = t3 – 88
7)   a[t4] = 0.0
8)   j = j + 1
9)   if j <= 10 goto (3)
10)  i = i + 1
11)  if i <= 10 goto (2)
12)  i = 1
13)  t5 = i – 1
14)  t6 = 88 * t5
15)  a[t6] = 1.0
16)  i = i + 1
17)  if i <= 10 goto (13)
```

= Primeira instrução

```
; Matriz identidade
; double a[10,10]

for i from 1 to 10 do
    for j from 1 to 10 do
        a[i,j] = 0.0;
for i from 1 to 10 do
    a[i,i] = 1.0;
```

# BBs: Exemplo

```
1)   i=1
2)   j=1
3)   t1=10 * i
4)   t2 = t1 + j
5)   t3 = 8 * t2
6)   t4 = t3 – 88
7)   a[t4] = 0.0
8)   j = j + 1
9)   if j <= 10 goto (3)
10)  i = i + 1
11)  if i <= 10 goto (2)
12)  i = 1
13)  t5 = i – 1
14)  t6 = 88 * t5
15)  a[t6] = 1.0
16)  i = i + 1
17)  if i <= 10 goto (13)
```

; Matriz identidade

; double a[10,10]

```
for i from 1 to 10 do
    for j from 1 to 10 do
        a[i,j] = 0.0;
for i from 1 to 10 do
    a[i,i] = 1.0;
```

= Primeira instrução = Alvo de Saltos

# BBs: Exemplo

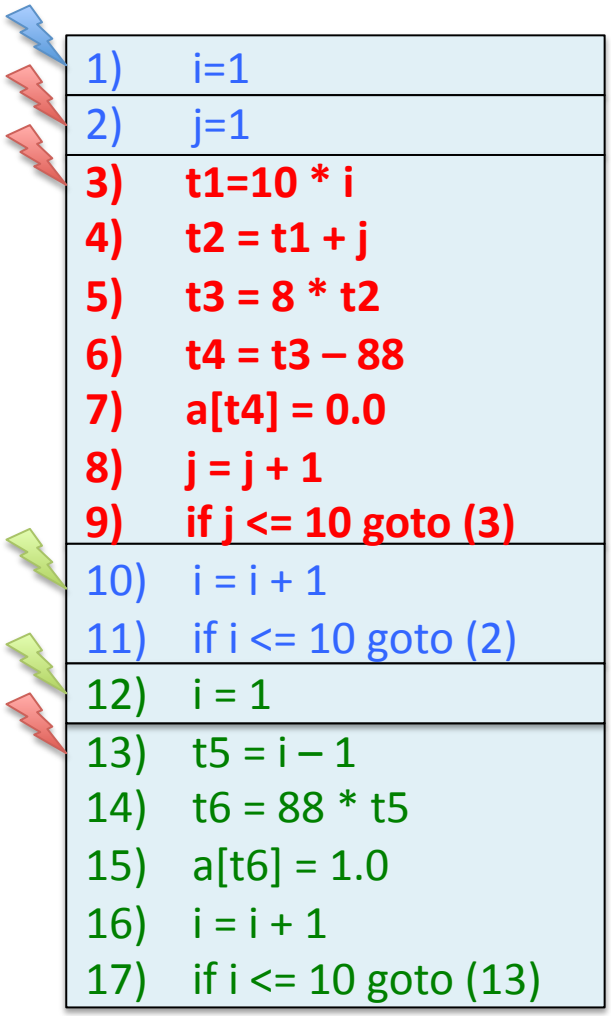
```
1)   i=1
2)   j=1
3)   t1=10 * i
4)   t2 = t1 + j
5)   t3 = 8 * t2
6)   t4 = t3 – 88
7)   a[t4] = 0.0
8)   j = j + 1
9)   if j <= 10 goto (3)
10)  i = i + 1
11)  if i <= 10 goto (2)
12)  i = 1
13)  t5 = i – 1
14)  t6 = 88 * t5
15)  a[t6] = 1.0
16)  i = i + 1
17)  if i <= 10 goto (13)
```

```
; Matriz identidade
; double a[10,10]

for i from 1 to 10 do
    for j from 1 to 10 do
        a[i,j] = 0.0;
for i from 1 to 10 do
    a[i,i] = 1.0;
```

= Primeira instrução = Alvo de Saltos = Sucessoras de saltos

# BBs: Exemplo




; Matriz identidade

; double a[10,10]

```
for i from 1 to 10 do
    for j from 1 to 10 do
        a[i,j] = 0.0;
for i from 1 to 10 do
    a[i,i] = 1.0;
```

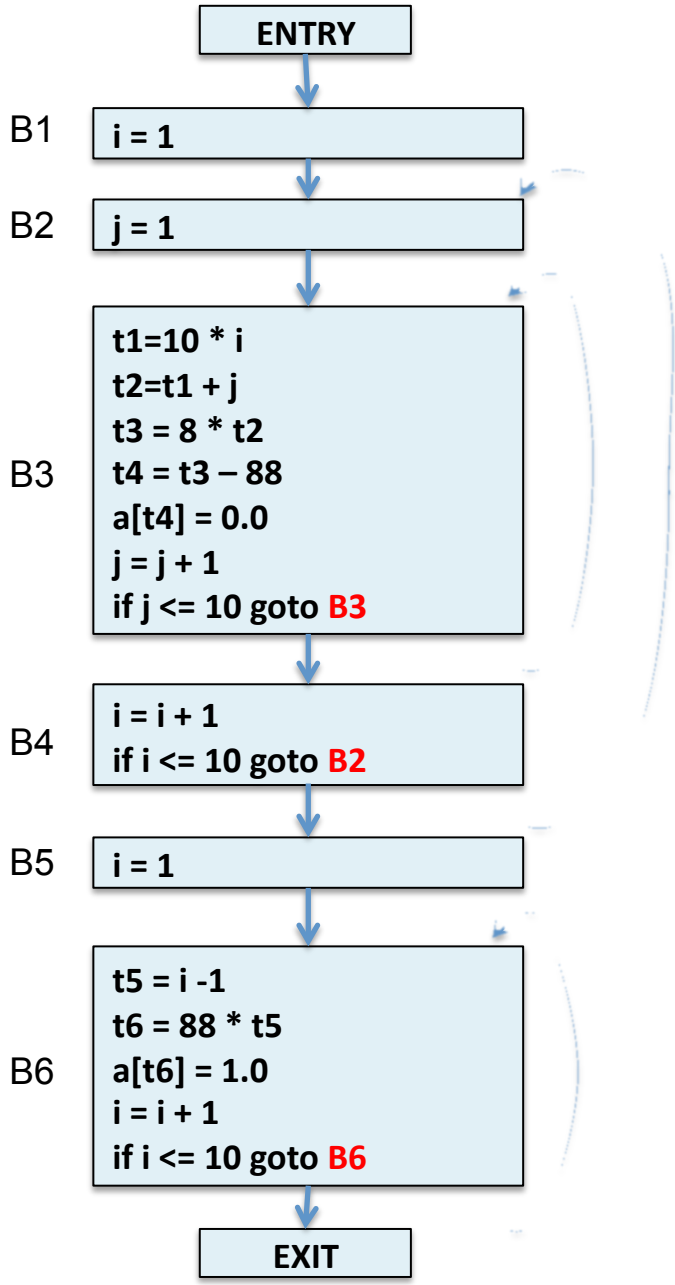
= Primeira instrução = Alvo de Saltos = Sucessoras de saltos

# CFG – Grafo de Fluxo de Controle

- Representação do fluxo de controle entre os blocos básicos.
- Para cada par de blocos B e C, há uma aresta B->C se e somente se a primeira instrução de C pode ser imediatamente executada após a última instrução de B. Isto acontece quando:
  - Há um salto condicional ou incondicional do final de B para o início de C.
  - C está posicionado logo após B na ordem original do programa e B não termina com um salto incondicional.

# CFG: Exemplo

| | | |
|---|---|---|
| B1 | 1) | i=1 |
| B2 | 2) | j=1 |
| B3 | 3) | **t1=10 * i** |
| | 4) | **t2 = t1 + j** |
| | 5) | **t3 = 8 * t2** |
| | 6) | **t4 = t3 – 88** |
| | 7) | **a[t4] = 0.0** |
| | 8) | **j = j + 1** |
| | 9) | **if j <= 10 goto (3)** |
| B4 | 10) | i = i + 1 |
| | 11) | if i <= 10 goto (2) |
| B5 | 12) | i = 1 |
| B6 | 13) | t5 = i – 1 |
| | 14) | t6 = 88 * t5 |
| | 15) | a[t6] = 1.0 |
| | 16) | i = i + 1 |
| | 17) | if i <= 10 goto (13) |



# Laços

Um laço é um conjunto *L,* de nós de um CFG, que contém um nó *e* (chamado de entrada do laço) tal que:

*1) e* não é o nó ENTRY

2) Nenhum nó em *L*, além de *e*, tem um predecessor fora de *L*. (Todo caminho do nó ENTRY a qualquer nó de *L* passa por *e*)

3) Todo nó em *L* tem um caminho não vazio completamente dentro de *L* para *e*.

# Laços

- Os laços do CFG visto anteriormente são:
  - B3
  - B6
  - {B2, B3, B4}
- Um laço que não contém outros laços é um laço interno (*inner loop*)

# Otimização de blocos básicos

# Directed Acyclic Graphs

- Úteis para transformações em BBs
  - Mostra como os valores computados são usados em sentenças subseqüentes
  - Common Sub-expression Elimination (CSE)
  - Dead Code Elimination
- Não confundir com o CFG
  - DAG: representa as operações de um BB
  - CFG: Nós são os BBs

Dragão – 8.5.1

# Directed Acyclic Graphs

- Construção:
  - Folhas são identificadores únicos
    - variáveis, constantes
    - são os valores iniciais das variáveis
    - usa-se índices para não confundir com valor atual

```
a = b + c
b = a – d
c = b + c
d = a – d
```

Exemplo: Dragão – 8.5.2

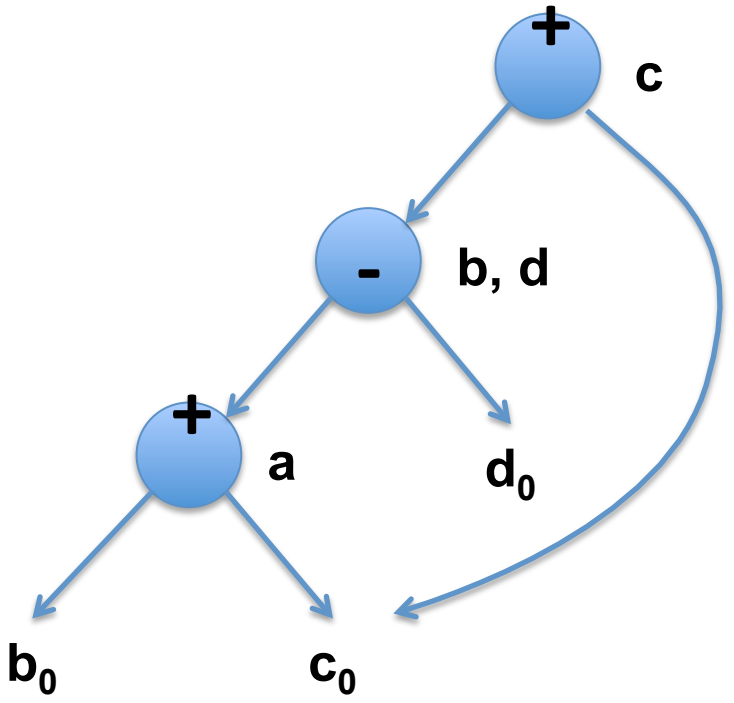
# Directed Acyclic Graphs

- Construção:
  - Nós internos são operadores:
    - valores computados
    - O rótulo é o operador associado
    - podem ter uma lista de variáveis associados
      - é o último valor computado para cada uma delas
    - um nó associado a cada sentença
    - filhos representam a última definição dos operandos

# DAGs – Exemplo 1

```
a = b + c
b = a – d
c = b + c
d = a – d
```

Dragão – 8.5.2

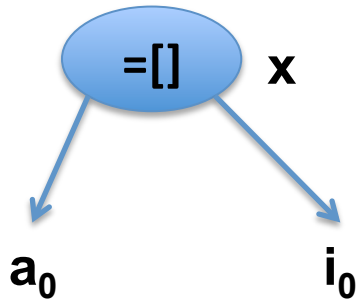


# DAGs – Exemplo 2

(1) x := a[i]

(2) a[j] := y

(3) z := a[i]

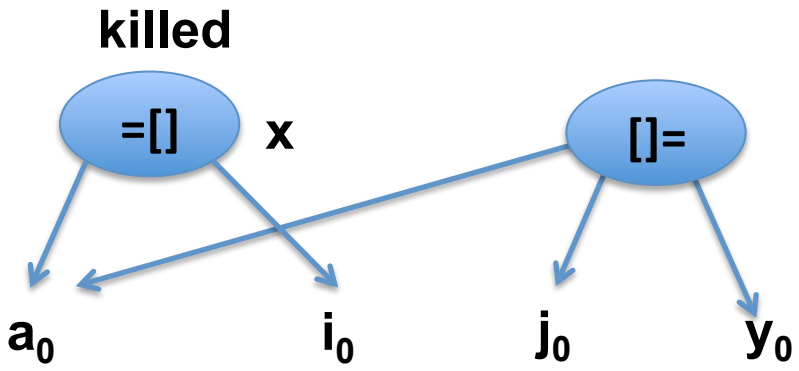


# DAGs – Exemplo 2

(1) x := a[i]

(2) a[j] := y

(3) z := a[i]

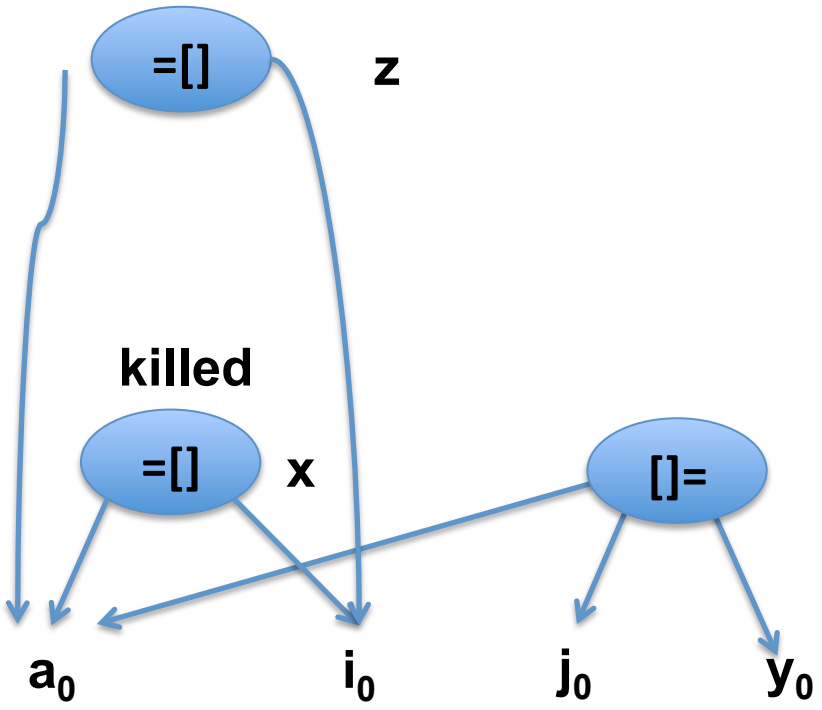


# DAGs – Exemplo 2

(1) x := a[i]

(2) a[j] := y

(3) z := a[i]



# Cuidados

- Arrays
  - `x := a[i]`
  - `a[j] := y`
  - `z := a[i]`
- Pode virar
  - `x := a[i]`
  - `z := x`
  - `a[j] := y`
- Ponteiros e chamada para funções, problema similar
  - `*p=w;` operador `*=` mata todos os nós do DAG
  - `call foo;` usa e altera todos os dados que tem acesso.

# Cuidados

- Ponteiros e chamada para funções, problema similar
  - `X =*p;` toma todos os nós como argumento
  - `*p=w;` operador *= mata todos os nós já construídos do DAG
  - `call foo;` usa e altera todos os dados que tem acesso.

# Aplicações

- Detectar sub-expressões comuns
  - automaticamente durante a construção
- Detectar os identificadores cujos valores são usados no bloco
  - São as folhas
- Detectar sentenças que geram valores que podem ser usados fora do bloco
  - São aquelas sentenças que geraram nó(x) = n durante a construção e ainda temos nó(x) =n ao final

# DAGs – Exemplo 3

```
(1) t1 := 4 * i
(2) t2 := a [t1]
(3) t3 := 4 * i
(4) t4 := b [t3]
(5) t5 := t2 * t4
(6) t6 := prod + t5
(7) prod := t6
(8)  t7 := i + 1
(9)  i := t7
(10) if i <= 20 goto (1)
```