MO615B - Implementação de Linguagens II
e
MC900A - Tópicos Especiais em Linguagem de Programação

Prof. Guido Araujo

www.ic.unicamp.br/~guido

Otimizações de Código

Introdução

- Usar as informações coletadas pelas análises
- Tornar o código mais eficiente
- Vamos começar olhando:
  - Dead Code Elimination
  - Constant Propagation
  - Copy Propagation
  - GCSE

Dead Code Elimination

- Se existem sentenças do tipo:
  - s:t = x + y
  - s:t = M[x]
  - De maneira que t não está vivo após s
    - Então s pode ser eliminada
- Instruções com efeito colateral
  - Podem provocar alteração no resultado do programa se forem removidas
    - Ex: overflow
  - O código pode funcionar com o otimizador ligado e não funcionar com ele ligado.

Dead Code Elimination

- Podemos
  - s:t = x + y
  - s:t = M[x]
  - De maneira que t não está vivo após s
    - Então s pode ser eliminada
- Para cada instrução com a du-chain vazia
  - Remova a instrução
  - Atualize a du-chain das definições que faziam parte da ud-chain dos usos da instrução .
    - Novas du-chains podem ficar vazias

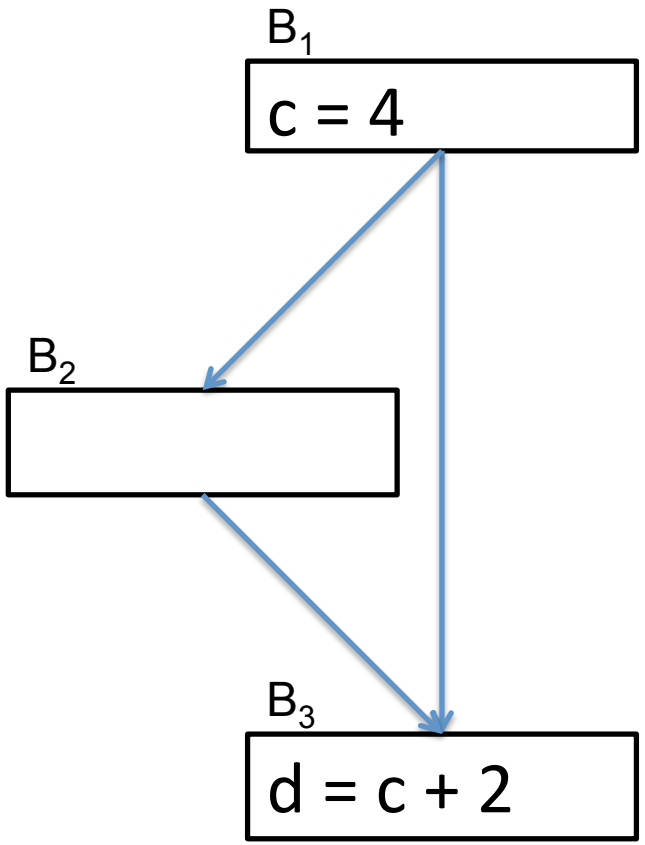
Dead Code Elimination

- Cuidado!
- Instruções com efeito colateral
  - Podem provocar alteração no resultado do programa se forem removidas
    - Ex: overflow
  - O código pode funcionar com o otimizador ligado e não funcionar com ele ligado.

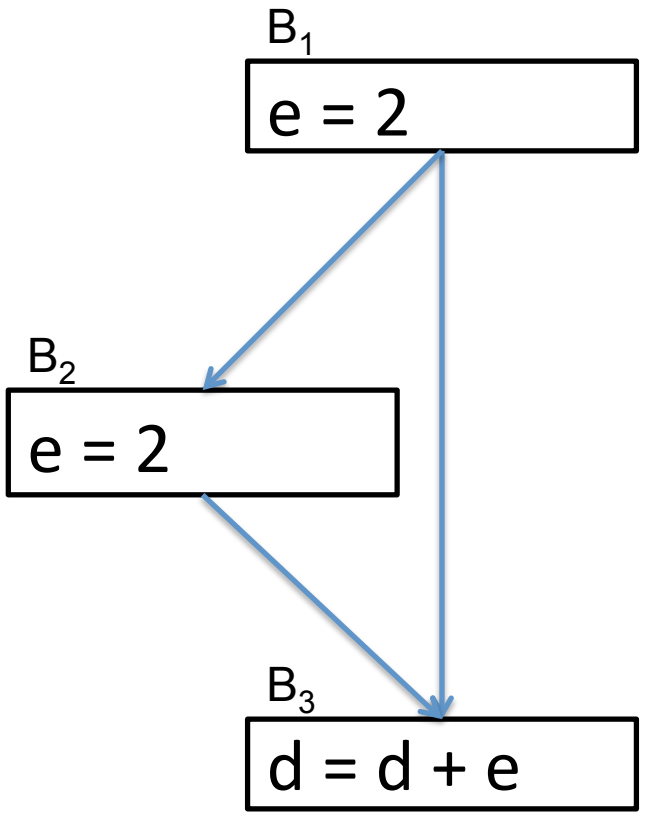
Constant Propagation

- Temos:
  - d: t = c , onde c é constante
  - n: y = t + x
- Sabemos que t é constante em n se
  - d alcança n
  - nenhuma outra definição de t alcança n
- Podemos reescrever
  - n: y = c + x

Constant Propagation - Exemplo



Constant Propagation - Exemplo



Constant Propagation

- No segundo exemplo não podemos aplicar a regra que diz que apenas uma definição d alcança n.
- Podemos modificar a regra para:
  - todas as definições d que alcançam n sejam cópias da mesma constante.

Constant Propagation

- Vamos assumir que durante nossa análise, em um determinado ponto do programa uma variável pode ser classificada como:
  - indefinida (UNDEF);
  - uma constant c;
  - não é uma constante (NAC).

Constant Propagation

- m é o conjunto de informações sobre as variáveis em um ponto do programa.

m(x) = UNDEF se x for classificada como indefinida
m(x) = NAC se x for classificada como uma não-constante.
m(x) = 2 se x for classificada como a constante 2

Constant Propagation

- no ponto após a sentença
  - s: x = 2
- Como é classificada a variável x?

Constant Propagation

- no ponto após a sentença
  - s: x = y + z
- Como é classificada a variável x?

Constant Propagation

  - s: x = y + z

- Assumindo que m e m’ são o conjunto de informações das variáveis antes e depois da sentença s, respectivamente, temos:

m’(x) = m(y) U m(z) se m(y) e m(z) forem constantes.
m’(x) = NAC se m(y) ou m(z) for NAC
m’(x) = UNDEF nos outros casos.

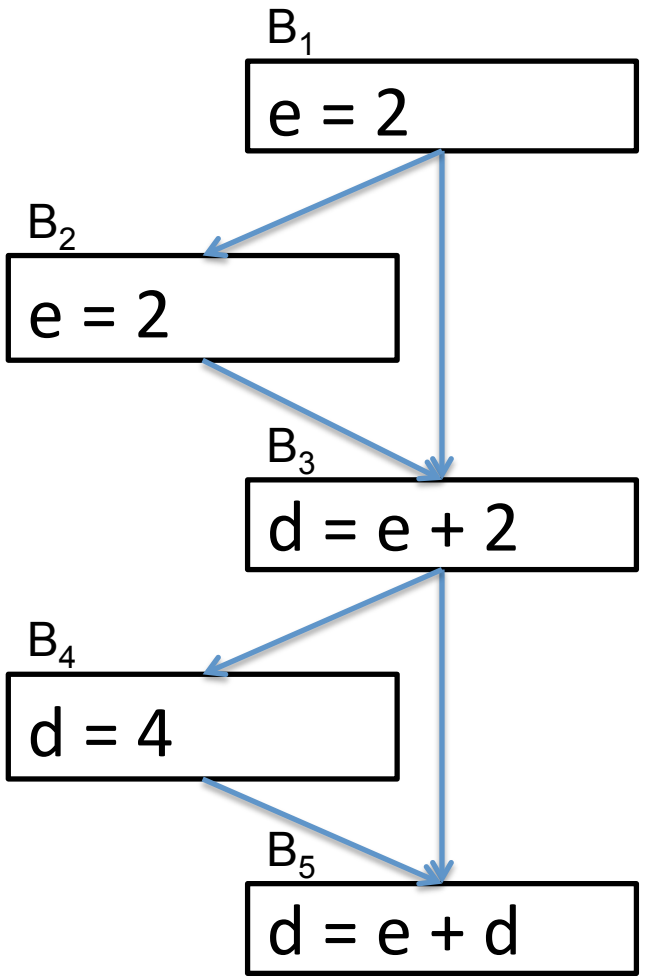
Constant Propagation

- seja IN[B] e OUT[B] os conjuntos m na entrada e saída do bloco B, respectivamente.

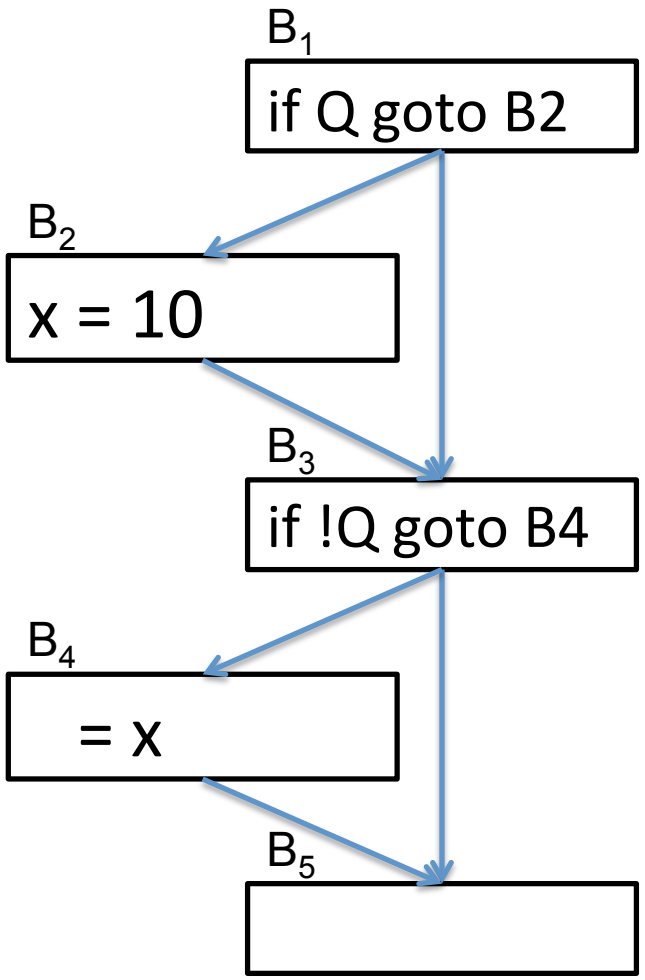
IN[B](x) =

- **const. c** *se existe um predecessor P' de B tal que*
  OUT[P'](x) = const.c e, para todo predecessor P de B,*
  OUT[P](x) = const. c ou OUT[P](x) = UNDEF
- **NAC** *se, para qualquer predecessor P de B,*
  OUT[P](x) = NAC
- **NAC** *se, para quaisquer par P1 e P2, predecessores de B,*
  OUT[P1](x) = c1 e OUT[P2](x) = c2, onde c1 != c2.
- **UNDEF** *nos outros casos*

Constant Propagation - Exemplo



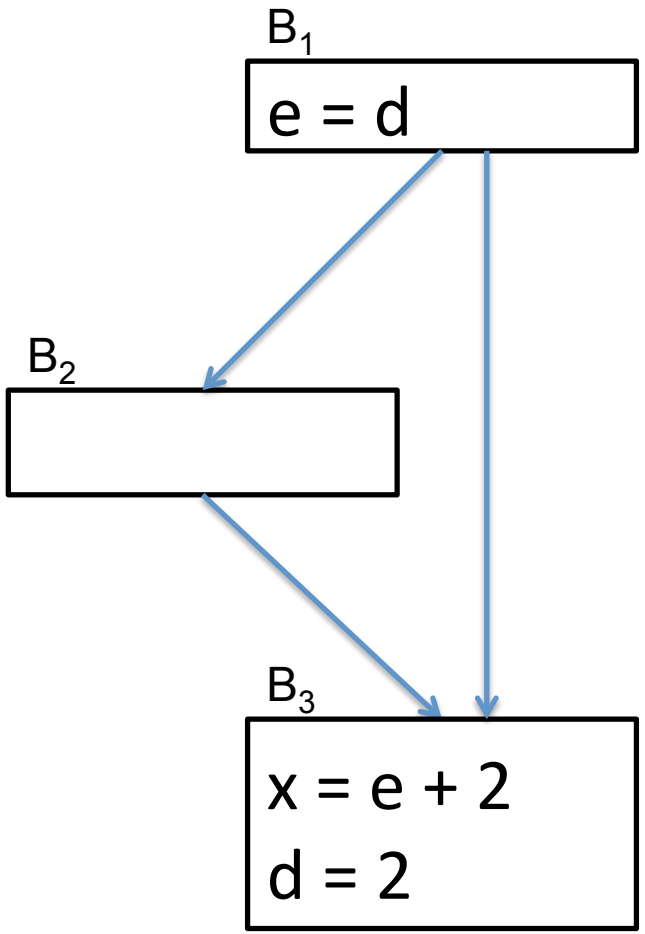
Constant Propagation - Exemplo



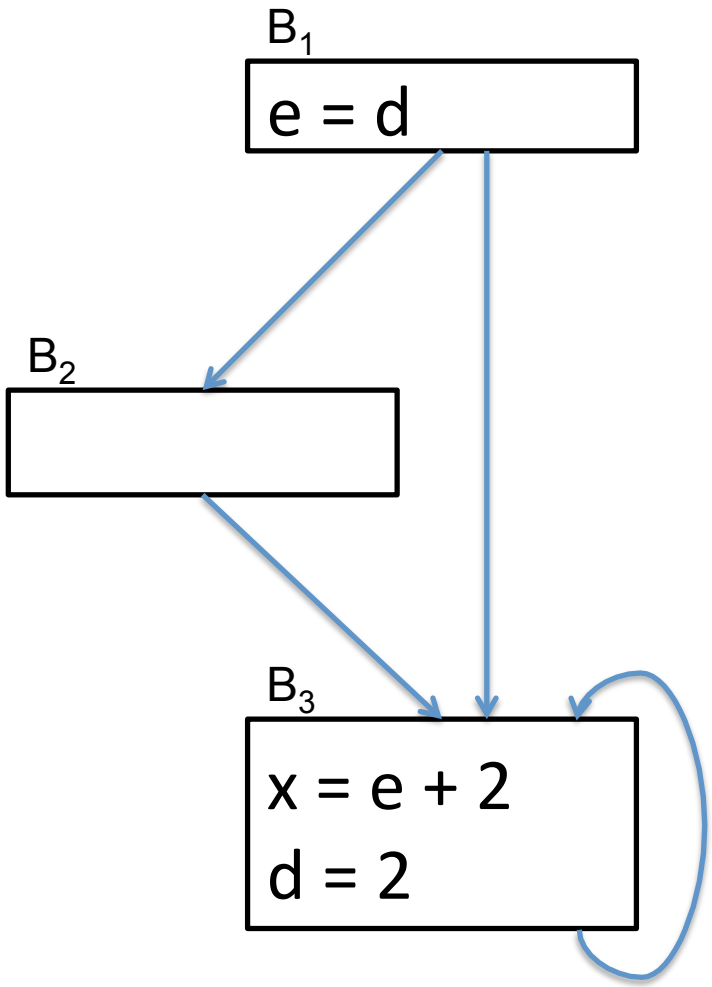
Copy Propagation

- Temos:
  - d: t = z , onde z é variável
  - n: y = t + x
- Sabemos que t é uma cópia em n se
  1. d alcança n
  2. nenhuma outra definição de t alcança n
  3. não existe definição de z em qualquer caminho de d a n, incluindo passagens sobre n uma ou mais vezes
- Podemos reescrever
  - n: y = z + x

Copy Propagation - Exemplo



Copy Propagation - Exemplo



Copy Propagation

1. d alcança n
2. nenhuma outra definição de t alcança n
   - Checadas usando ud-chains

3. não existe definição de z em qualquer caminho de d a n, incluindo passagens sobre n uma ou mais vezes
   - Nova *dataflow analysis*
     - c_in[B]: conjunto de cópias s: x = y tais que todo caminho do início até o nó B contém a sentença s, e após a última ocorrência de s não há atribuições a y
     - c_out[B]: idem para o final de B

Copy Propagation

- Condição 3
  - Nova *dataflow analysis*
    - c_gen[B]: s ocorre em B e não há atribuição a y após s
    - c_kill[B]: s é morta em B se x ou y são atribuídos em B e s não está em B
      - Note que diferentes atribuições x = y matam umas às outras
      - c_in[B] só pode conter uma sentença x = y com x à esquerda

Copy Propagation

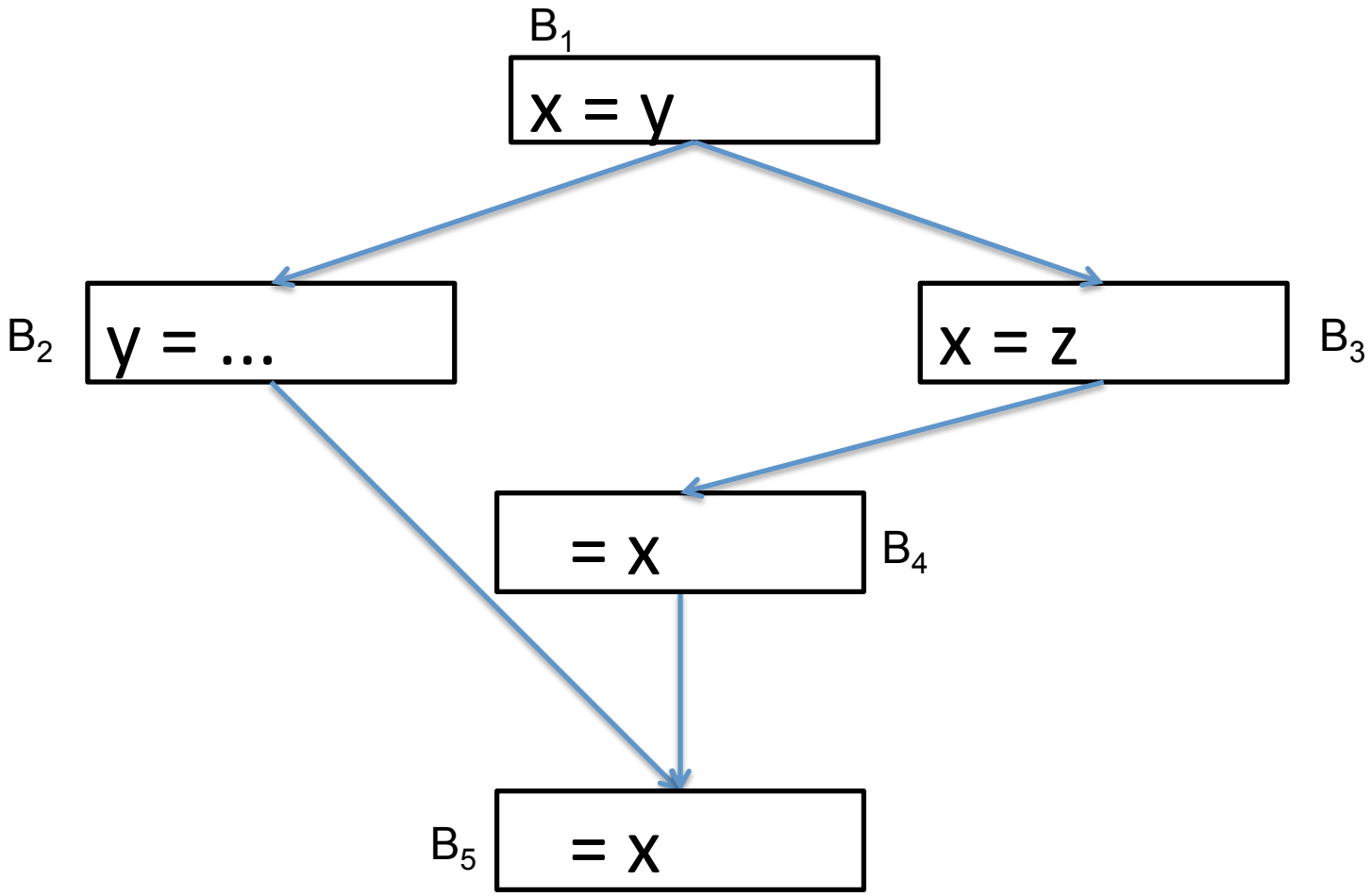
- Condição 3
  - Equações
  - As mesmas de Expressões disponíveis!
    - É chamada de Cópias Disponíveis

$$in[B] = \bigcap_{P \in Pred(B)} out[P]$$

$$in[B1] = \varnothing$$

$$out[B] = c\_gen[B] \bigcup (in[B] - c\_kill[B])$$

Copy Propagation - Exemplo



Exemplo

- Teremos:
  - c_gen[B1] = {x=y}; c_kill[B1] = {x=z}
  - c_gen[B2] = {}; c_kill[B2] = {x=y}
  - c_gen[B3] = {x=z}; c_kill[B1] = {x=y}
  - Os outros são vazios
- Resolvendo:
  - In[B1] = {}; In[B2] = in[B3] = out[B1] = {x=y}
  - Out[B2]={}
  - Out[B3] = in[B4] = out[B4] = {x=z}
  - In[B5] = out[B2] $\cap$ out[B4] = {}

Copy Propagation Optimization

Algorithm 10.6. Copy propagation.

Input. A flow graph G, with ud-chains giving the definitions reaching block B, and with $c\_in[B]$ representing the solution to Equations 10.12, that is, the set of copies `x:=y` that reach block B along every path, with no assignment to `x` or `y` following the last occurrence of `x:=y` on the path. We also need du-chains giving the uses of each definition.

Output. A revised flow graph.

Method. For each copy s: `x:=y` do the following.

1. Determine those uses of `x` that are reached by this definition of `x`, namely, s: `x:=y`.
2. Determine whether for every use of `x` found in (1), s is in $c\_in[B]$, where B is the block of this particular use, and moreover, no definitions of `x` or `y` occur prior to this use of `x` within B. Recall that if s is in $c\_in[B]$, then s is the only definition of `x` that reaches B.
3. If s meets the conditions of (2), then remove s and replace all uses of `x` found in (1) by `y`. □

GCSE

- Eliminar as expressões comuns
  - Usa “Expressões Disponíveis”
- Vejamos o algoritmo:

GCSE

Algorithm 10.5. Global common subexpression elimination.

Input. A flow graph with available expression information.

Output. A revised flow graph.

Method. For every statement s of the form `x:=y+z`[6] such that `y+z` is available at the beginning of s's block, and neither `y` nor `z` is defined prior to statement s in that block, do the following.

1. To discover the evaluations of `y+z` that reach s's block, we follow flow graph edges, searching backward from s's block. However, we do not go through any block that evaluates `y+z`. The last evaluation of `y+z` in each block encountered is an evaluation of `y+z` that reaches s.
2. Create a new variable `u`.
3. Replace each statement `w:=y+z` found in (1) by

```
u := y+z
w := u
```

4. Replace statement s by `x:=u`. □

Exemplo 1

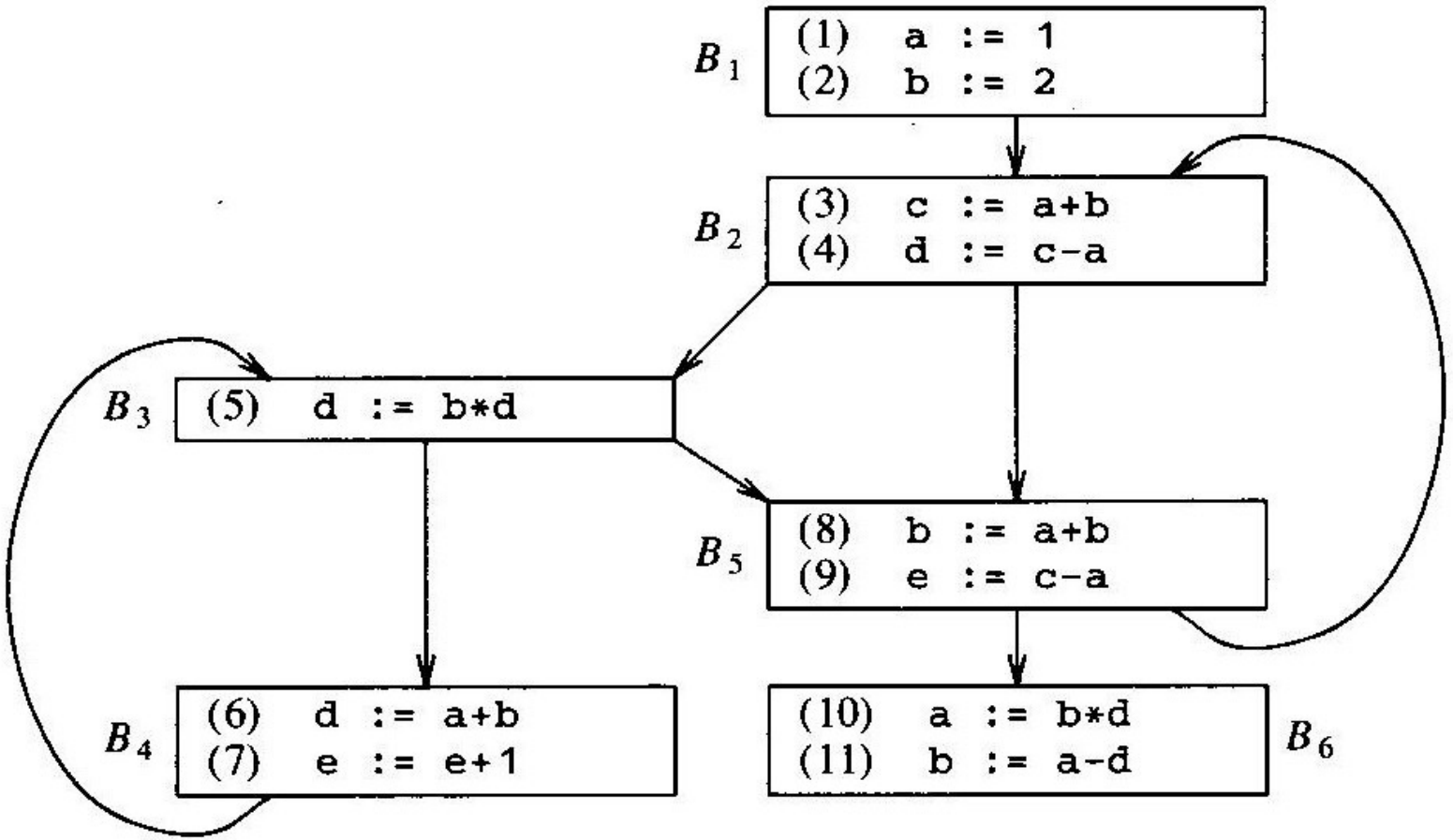
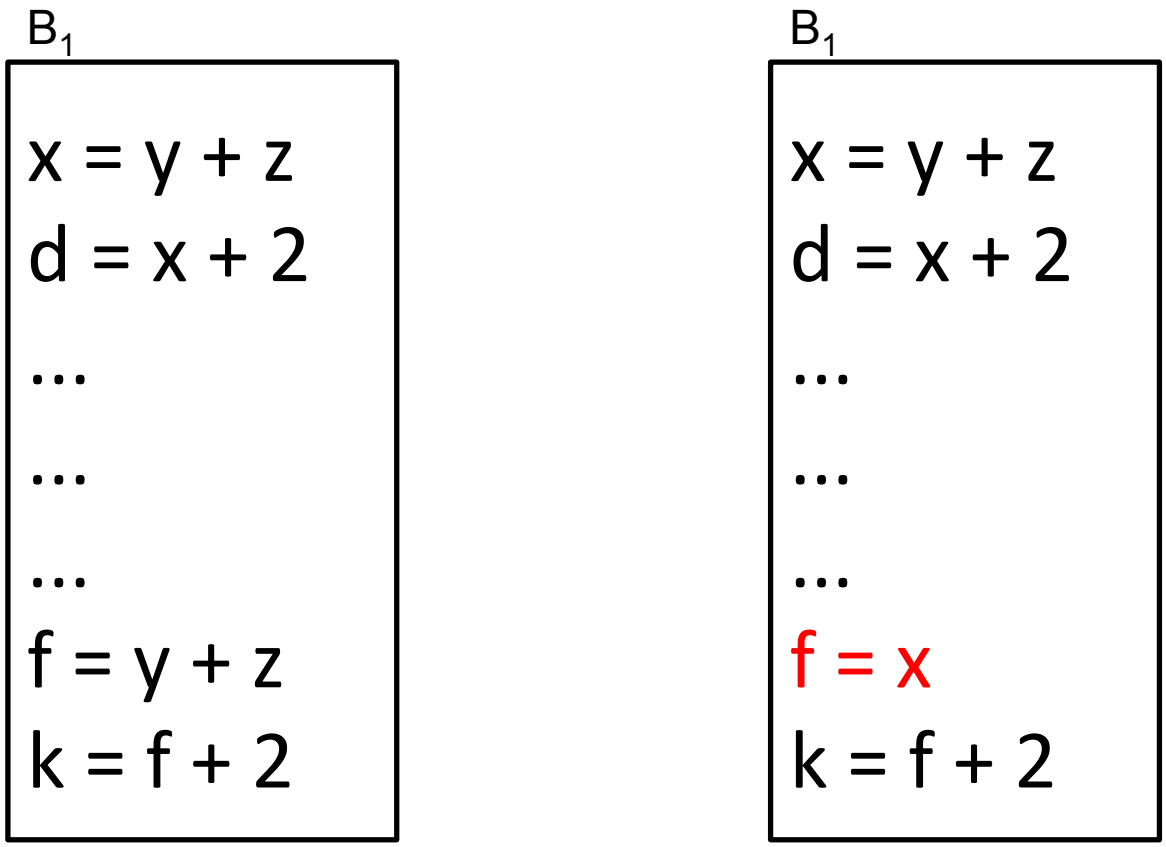


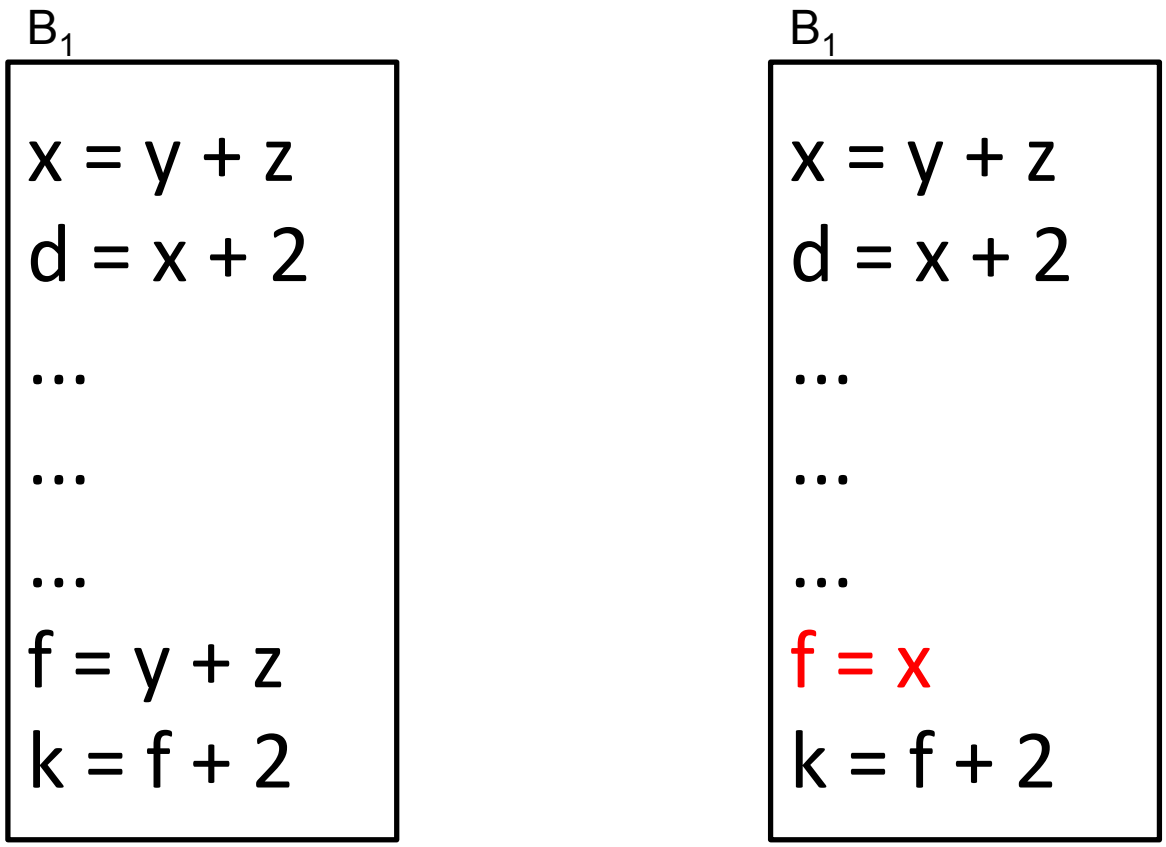
Fig. 10.74. Flow graph.

Exemplo 2



- O código ficou melhor?

Exemplo 2



- O que acontece com o *live-range* da variável após GCSE?