Transformações para Otimização de Código Parte I

Sandro Rigo
sandro@ic.unicamp.br

Introdução

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

Introdução

- Transformações Globais
  - Não substituem as locais
  - Ambas devem ser aplicadas

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

Copy Propagation

- Temos:
  - d: t = z , onde z é variável
  - n: y = t + x
- Sabemos que t é constante 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

- Condições 1 e 2
  - Checadas usando ud-chains
- Condição 3
  - 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 as 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!

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

$$in[B1] = \varnothing$$

$$out[B] = gen[B] \bigcup (in[B] - kill[B])$$

Exemplo

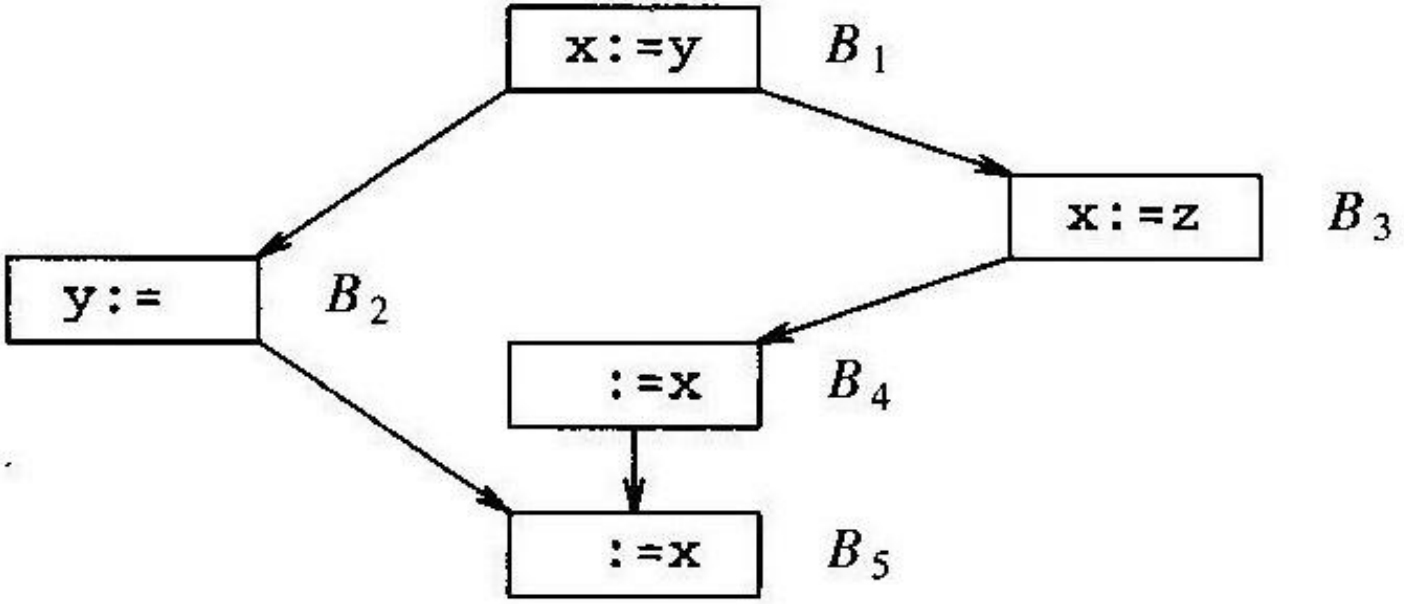

Fig. 10.35. Example flow graph.

Copy Propagation

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

- Determinar 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`. □

GCSE

- Passo 1:
  - Pode ser formulado como uma DFA, conhecida como Reaching Expressions
    - t = x + y (em um nó N do CFG) alcança um nó S:
      - Se existe um caminho de N a S sem atribuições a x ou y, ou computação de x+y
  - Não é feito dessa maneira pela quantidade de informação inútil que iria coleta
    - É necessária apenas para algumas expressões

Exemplo

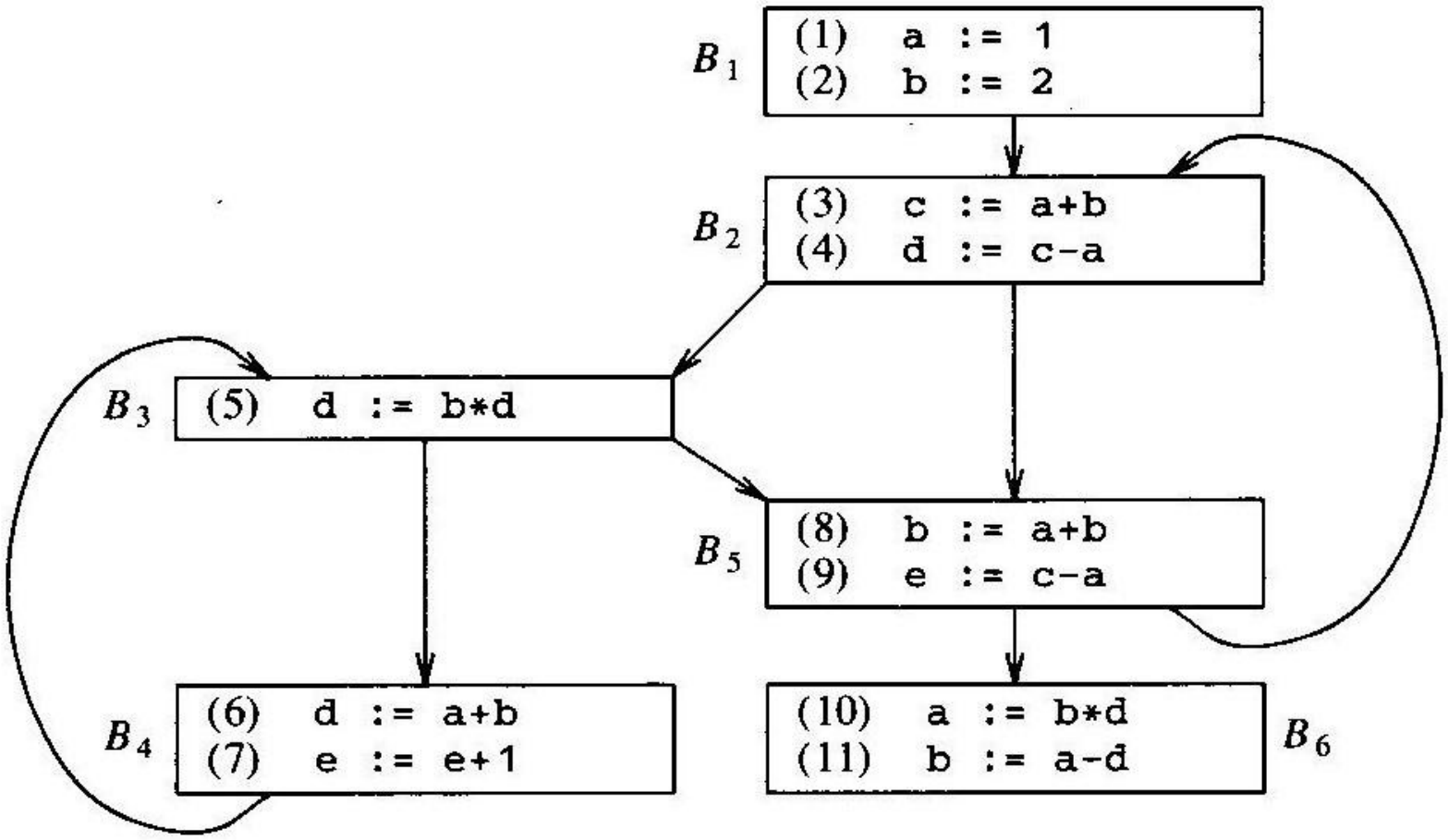

Fig. 10.74. Flow graph.

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.