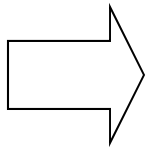
CSE

CSE – Common Subexpression Elimination

- Determinar as expressões comuns
  - Usa “Expressões Disponíveis”

- Eliminar recomputos de expressões:

```
i = a + b
j = a + b
```

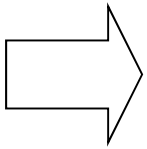


```
u = a + b
i = u
j = u
```

CSE

- Porque:

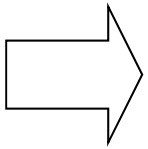
```
i = a + b
j = a + b
```



```
u = a + b
i = u
j = u
```

- Ao invés de:

```
i = a + b
j = a + b
```



```
i = a + b
j = i
```

CSE

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`.

CSE

- 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 coletar
    - É necessária apenas para algumas expressões

Análise de Fluxo de Dados: Available Expressions

Available Expressions

- **Expressão disponível**:

  $x+y$ está disponível em um ponto $\boldsymbol{p}$ se:

  - todo caminho do nó inicial até $\boldsymbol{p}$ calcula $x+y$
  - após a última computação de $x+y$, nem x nem y sofrem atribuições

- **kill**:
  - Um bloco B mata, ou pode matar, $x+y$ se ele atribui a x e/ou y, e não recomputa $x+y$

- **gen**:
  - Um bloco B gera $x+y$ se ele certamente computa $x+y$, e não redefine x ou y.

Available Expressions: gen e kill

STATEMENTS	AVAILABLE EXPRESSIONS
.....	none
a := b+c	
.....	only b+c
b := a-d	
.....	only a-d
c := b+c	
.....	only a-d
d := a-d	
.....	none

Fig. 10.30. Computation of available expressions.

Available Expressions: Equações de DFA

- Computa-se gen e kill para cada B.
- Tem-se:

$$in[B1] = \varnothing$$

$$in[B] = \bigcap_{P \in \Pr ed(B)} out[P] \quad \textit{para B não inicial}$$

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

Available Expressions: Diferenças com Reaching Definitions

- O ***in*** do nó inicial é sempre vazio
  - Nada está disponível antes do início do programa

- O operador de confluência é intersecção
  - Tem que vir por todos os caminhos

- Estimativa inicial é muito grande
  - Intersecção vai diminuindo os conjuntos a chegar ao maior ponto fixo

Available Expressions: Solução Iterativa

Algorithm 10.3. Available expressions.

Input. A flow graph G with $e\_kill[B]$ and $e\_gen[B]$ computed for each block B. The initial block is B_1.

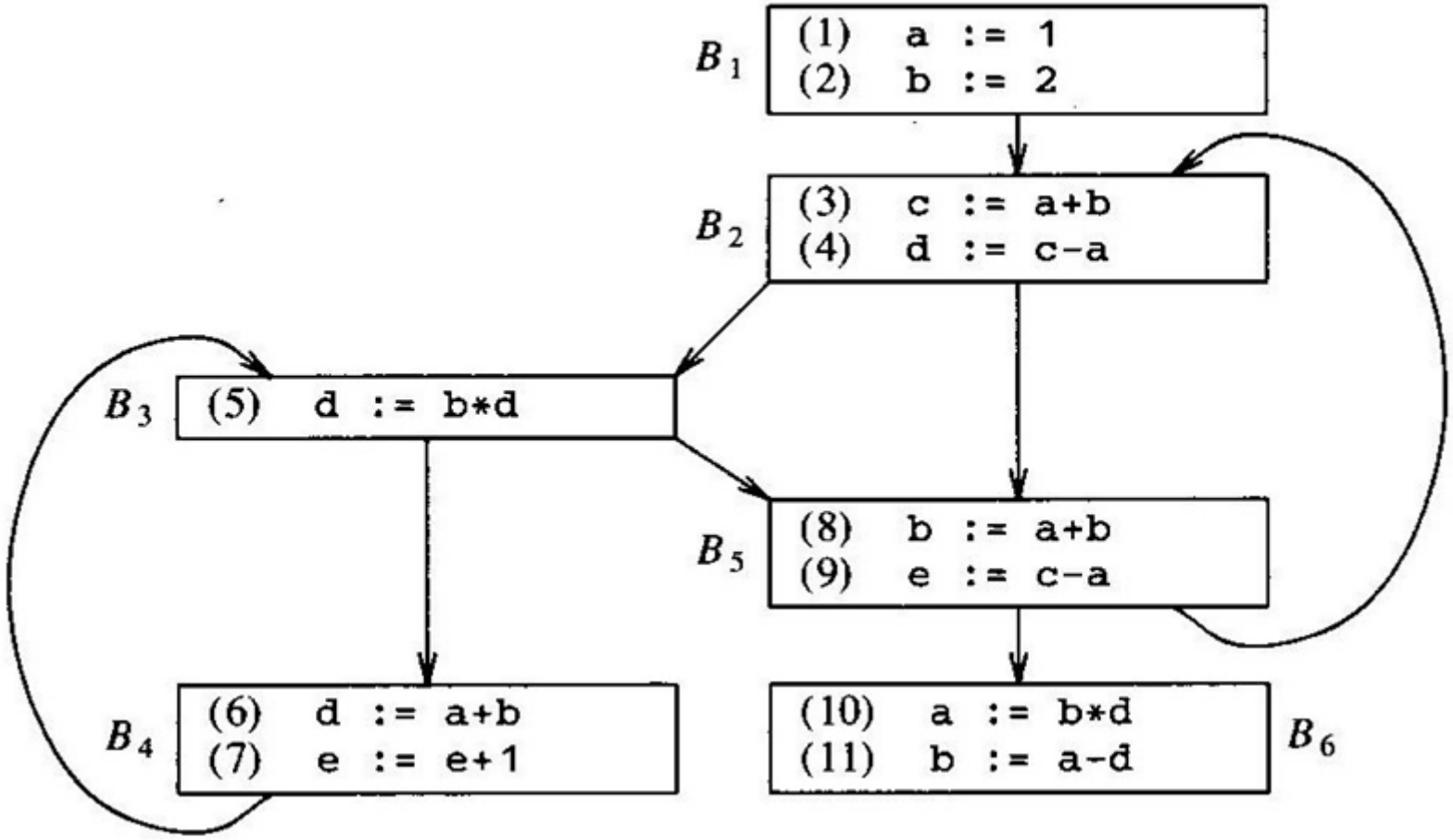
Output. The set $in[B]$ for each block B.

Method. Execute the algorithm of Fig. 10.32. The explanation of the steps is similar to that for Fig. 10.26. □

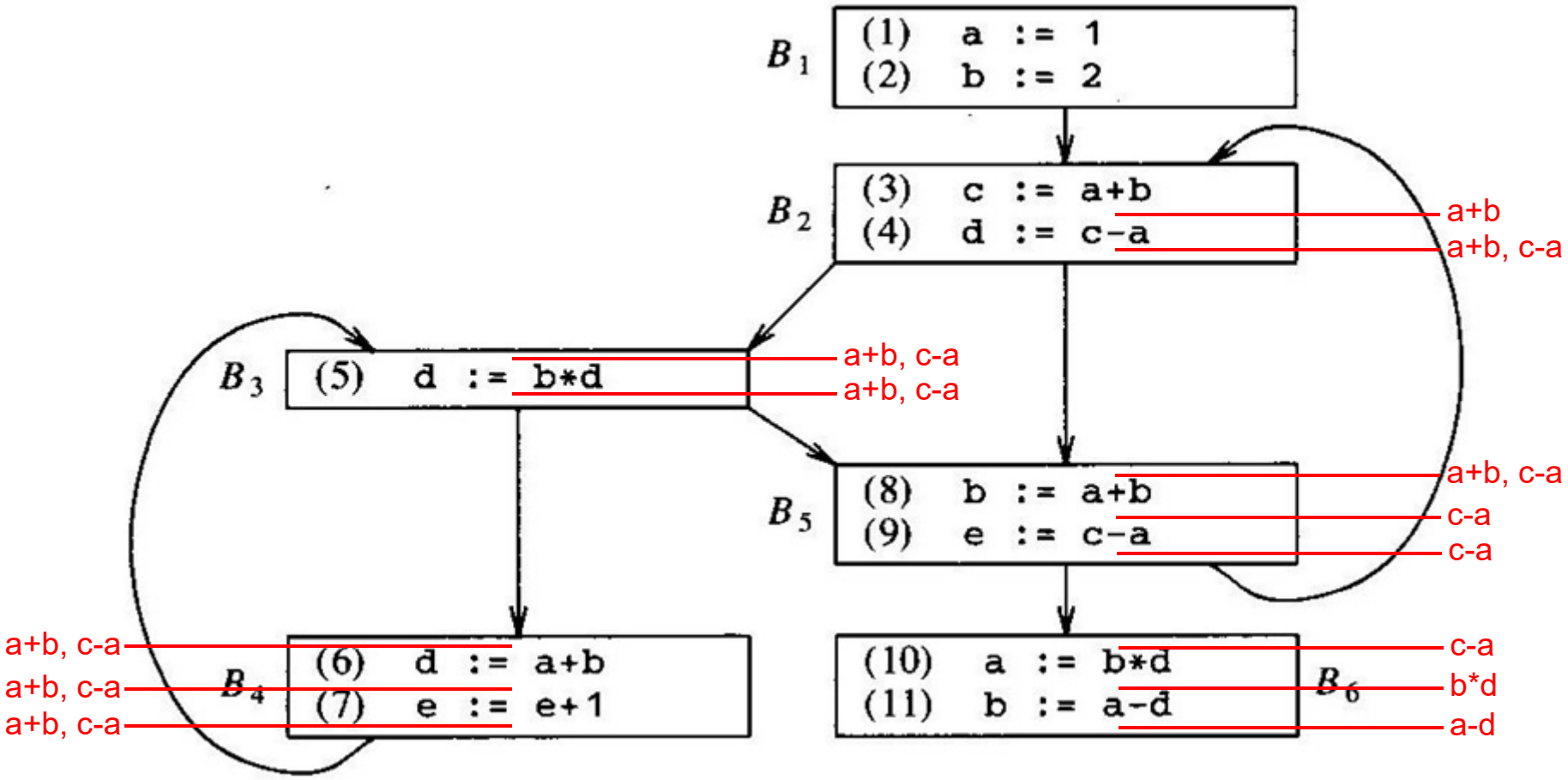
```
in[B1] := ∅;
out[B1] := e_gen[B1];  /* in and out never change for the initial node, B1 */
for B ≠ B1 do out[B] := U − e_kill[B];  /* initial estimate is too large */
change := true;
while change do begin
        change := false;
        for B ≠ B1 do begin
                in[B] := ⋂ out[P];
                         P a predecessor of B
                oldout := out[B];
                out[B] := e_gen[B] ∪ (in[B] − e_kill[B]);
                if out[B] ≠ oldout then change := true
        end
end
```

Fig. 10.32. Available expressions computation.

Available Expressions: Exemplo



CSE: Exemplo


B1
(1) a := 1
(2) b := 2
B2
(3) c := a+b
(4) d := c-a
a+b
a+b, c-a
B3
(5) d := b*d
a+b, c-a
a+b, c-a
B5
(8) b := a+b
(9) e := c-a
a+b, c-a
c-a
c-a
B4
(6) d := a+b
(7) e := e+1
a+b, c-a
a+b, c-a
a+b, c-a
B6
(10) a := b*d
(11) b := a-d
c-a
b*d
a-d