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

Prof. Guido Araujo

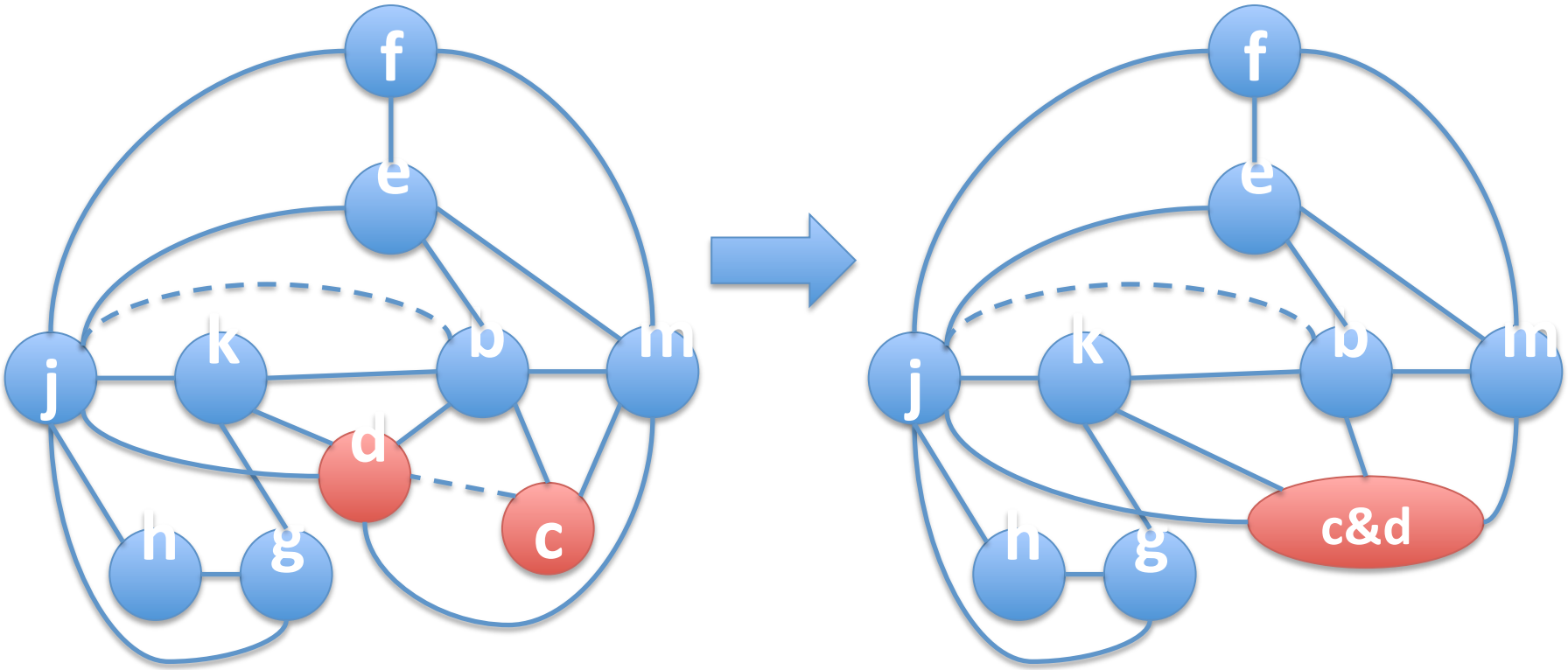
www.ic.unicamp.br/~guido

Coalescing

Coalescing

- Eliminar cópias (MOVES) redundantes usando o IG.
  - Se não existirem arestas entre os nós das variáveis de uma instrução de cópia ela pode ser eliminada.
- Os nós fonte e destino da cópia são unidos (*coalesced*) em um só nó.
- O conjunto de arestas do novo nó é a união das arestas dos dois anteriores.

Exemplo



Coalescing

- O efeito é sempre benéfico?
  - Qualquer instrução de cópia sem arestas no IG poderia ser eliminada
  - Pode tornar o processo de alocação mais complicado
    - Por que?

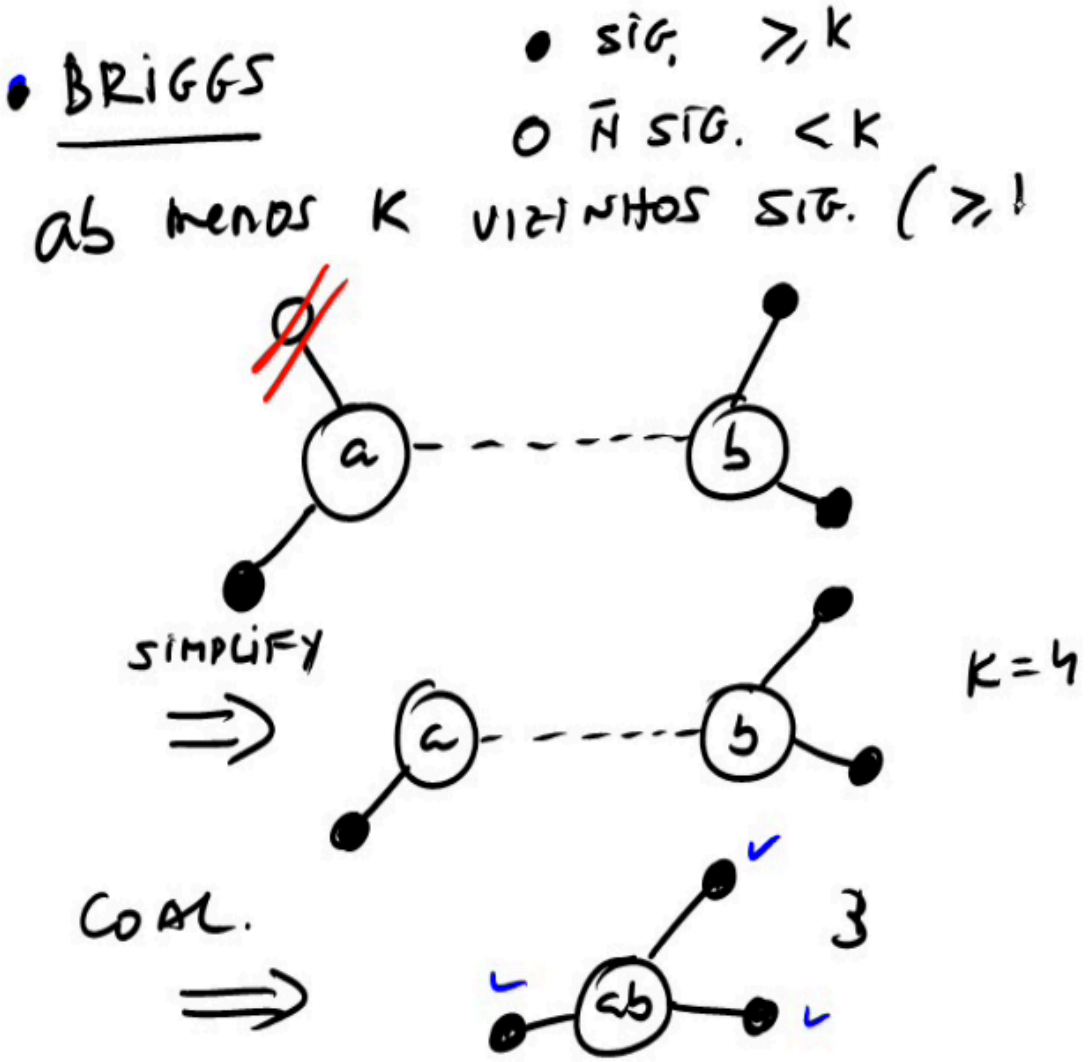
Coalescing

- O efeito é sempre benéfico?
  - Qualquer instrução de cópia sem arestas no IG poderia ser eliminada
  - Pode tornar o processo de alocação mais complicado
    - Por que?
- O nó resultante é mais restritivo que os anteriores
  - Seu grau aumenta.
  - Pode se tornar >= K.
- Um grafo k-colorível antes do *coalescing* pode se tornar não k-colorível após uma operação de *coalescing*

Coalescing

- Devemos tomar cuidados
  - Executar *coalescing* somente quando for seguro.
  - Temos duas estratégias: Briggs e George.
- Briggs:
  - *a* e *b* podem ser unidos se o nó resultante *ab* tiver menos do que *K* vizinhos com grau significativo (>=*K*)
  - Garante que o grafo continua *k*-colorível. Por quê?
  - Após a simplificação remover todos os nós não-significativos, sobram menos do que K vizinhos para o nó *ab.*
  - Logo, ele pode ser removido.

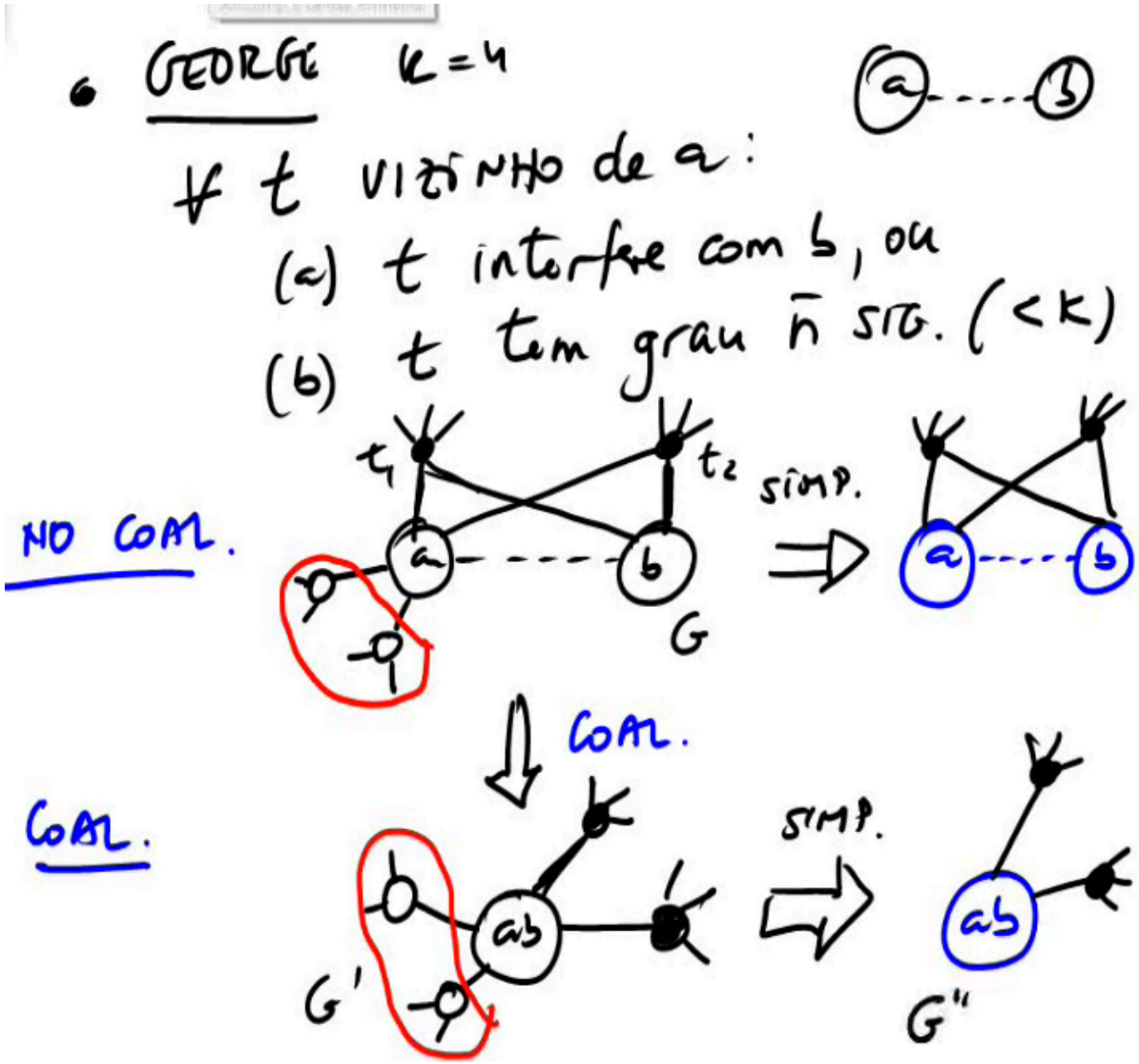
Heurística de Briggs



Coalescing

- George:
  - *a* e *b* podem ser unidos se para cada vizinho *t* de *a:*
    - *t* interfere com *b*
    - ou *t* tem grau insignificante (<K)
  - Por que é segura?

Heurística de George



Coalescing

- George:
  - *a* e *b* podem ser unidos se para cada vizinho *t* de *a:*
    - *t* interfere com *b*
    - ou *t* tem grau insignificante (<K)
  - Por que é segura?
  - Seja S o conjunto de vizinhos insignificantes de a em G
  - Sem o coalescing, todos poderiam ser removidos, gerando um grafo G1
  - Fazendo o coalescing, todos os nós de S também poderão ser removidos, criando G2
  - G2 é um subgrafo de G1, onde o nó *ab* corresponde ao *b*
  - *G2* é no mímino tão fácil para colorir quanto G1

Coalescing

- São estratégias conservativas
- Podem sobrar operações de cópia que poderiam ser removidas
- Ainda assim, é melhor do que fazer *spill*!

Fases da Alocação com *Coalescing*

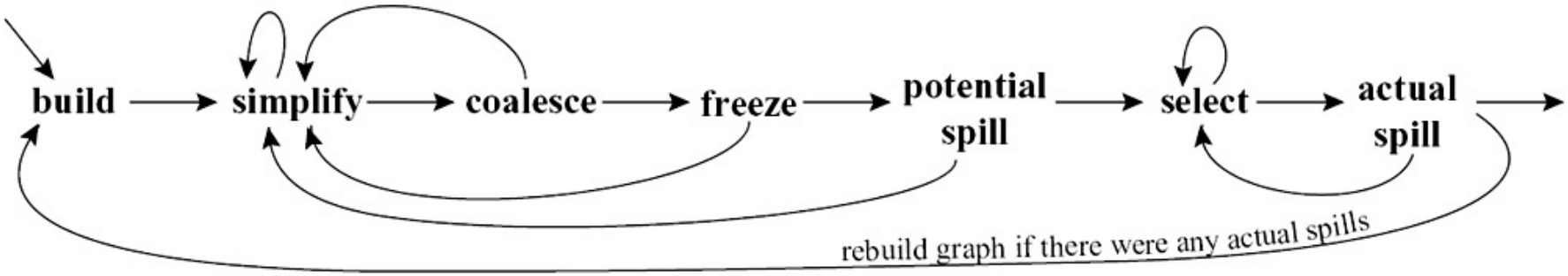
- *Build*:
  - Construir o IG.
  - Categorizar os nós em *move-related* e *move-unrelated.*
- *Simplify*:
  - Remover os nós não significativos (grau < K) que são ***move-unrelated***, um de cada vez.
- *Coalesce*:
  - Faça o coalesce conservativo no grafo resultante do passo anterior.
  - Com a redução dos graus, é provável que apareça mais oportunidades para o *coalescing*.
  - Quando um nó resultante não é mais *move-related* ele fica disponível para a próxima simplificação.

Fases da Alocação com *Coalescing*

- *Freeze*:
  - Executado quando nem o *simplify* nem o *coalescing* podem ser aplicados
  - Procura nós *move-related* de grau baixo.
  - Congela os moves desses nós. Eles passam a ser candidatos para simplificação.
- *Spill*:
  - Se não houver nós de grau baixo, selecionamos um nó com grau significativo para spill.
  - Coloca-se esse nó na pilha.
- *Select*:
  - Desempilhar todos os nós e atribuir cores.

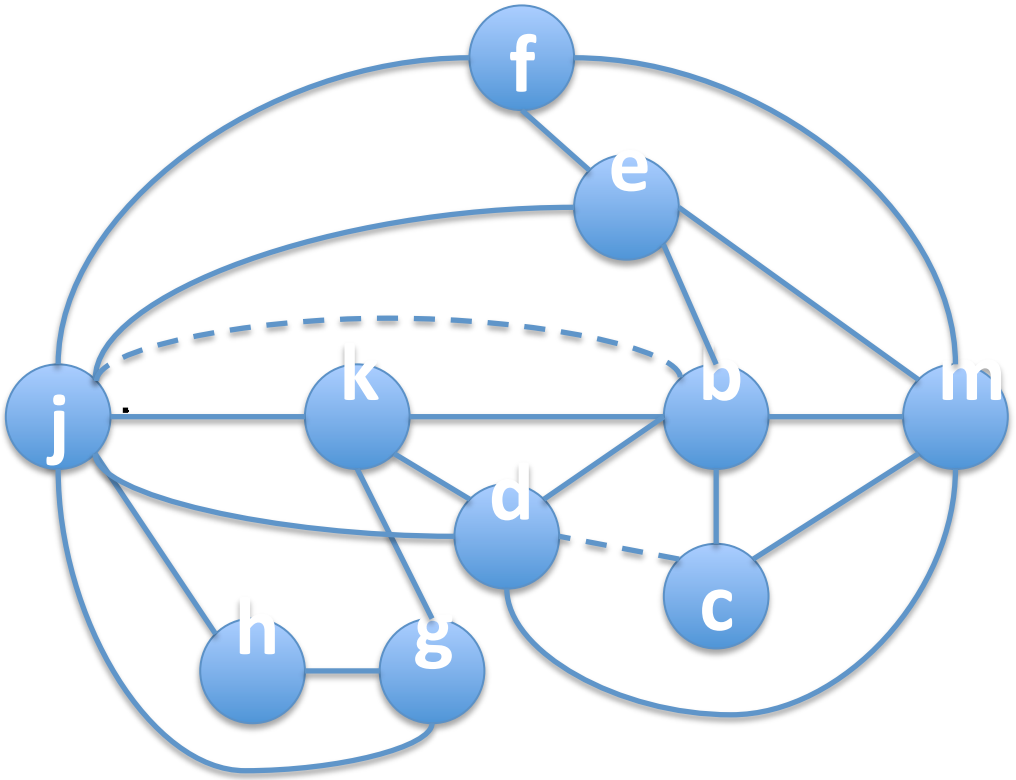
Fluxo com Coalescing

- *Simplify*, *coalesce* e *spill* são intercalados até que o grafo esteja vazio.



Retomando o Exemplo (K=4)

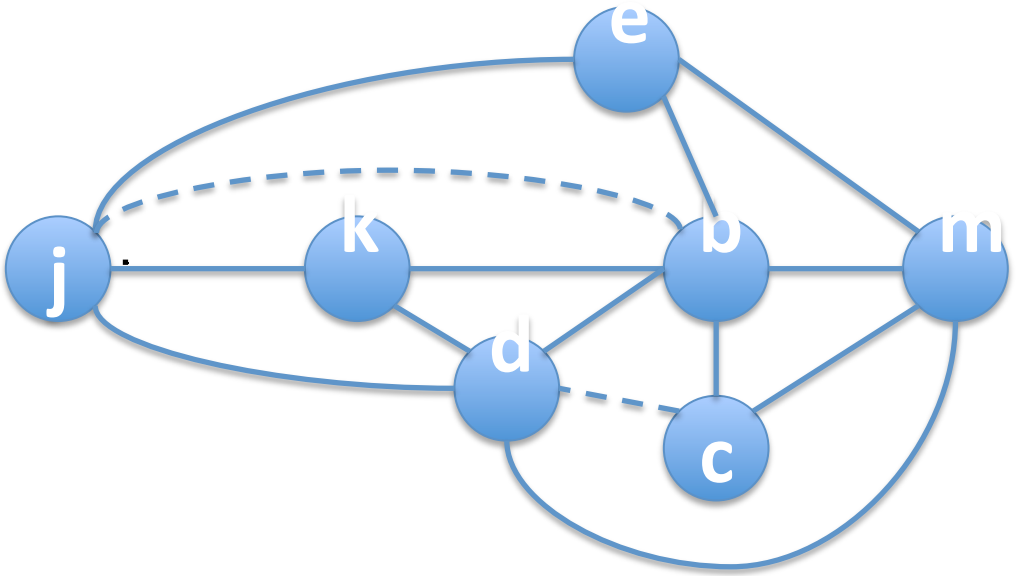
- Removendo h, g , f



- Agora somente nós não relacionados a cópias podem ser candidatos no *simplify*

Retomando o Exemplo

- Agora podemos remover: e, k



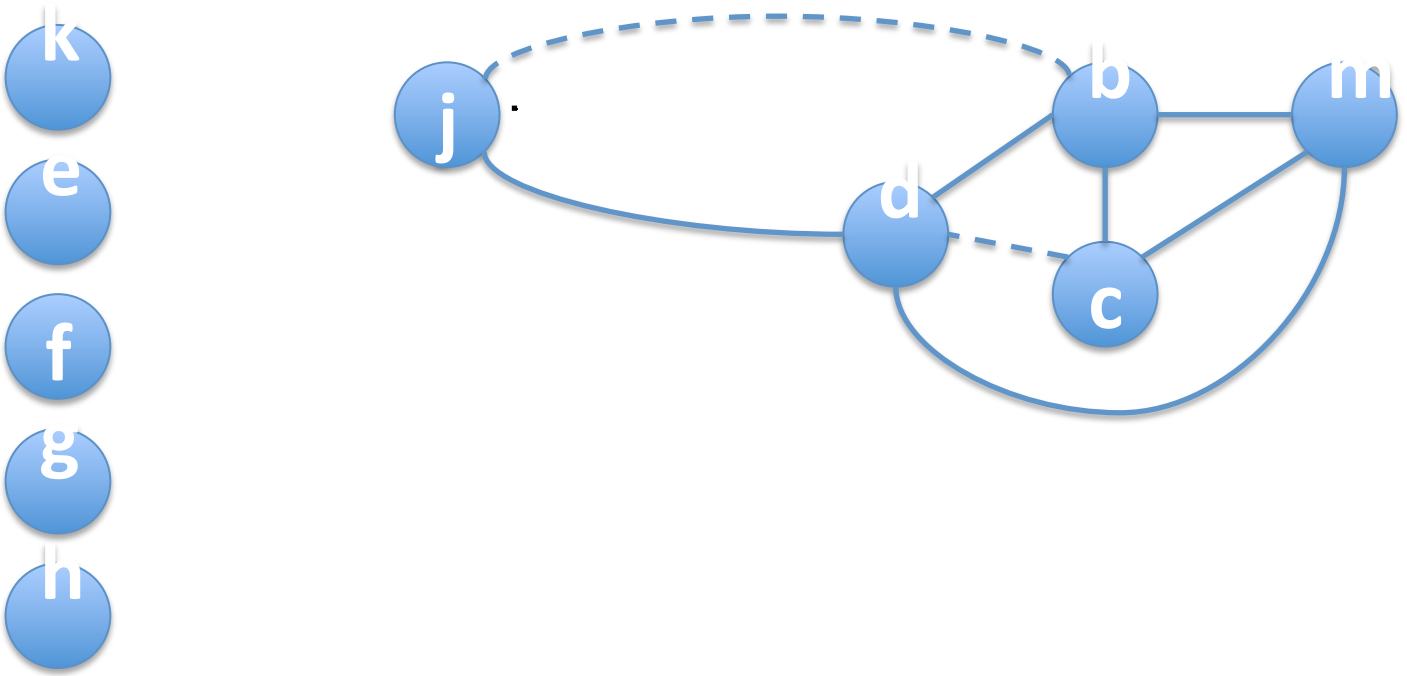
f

g

h

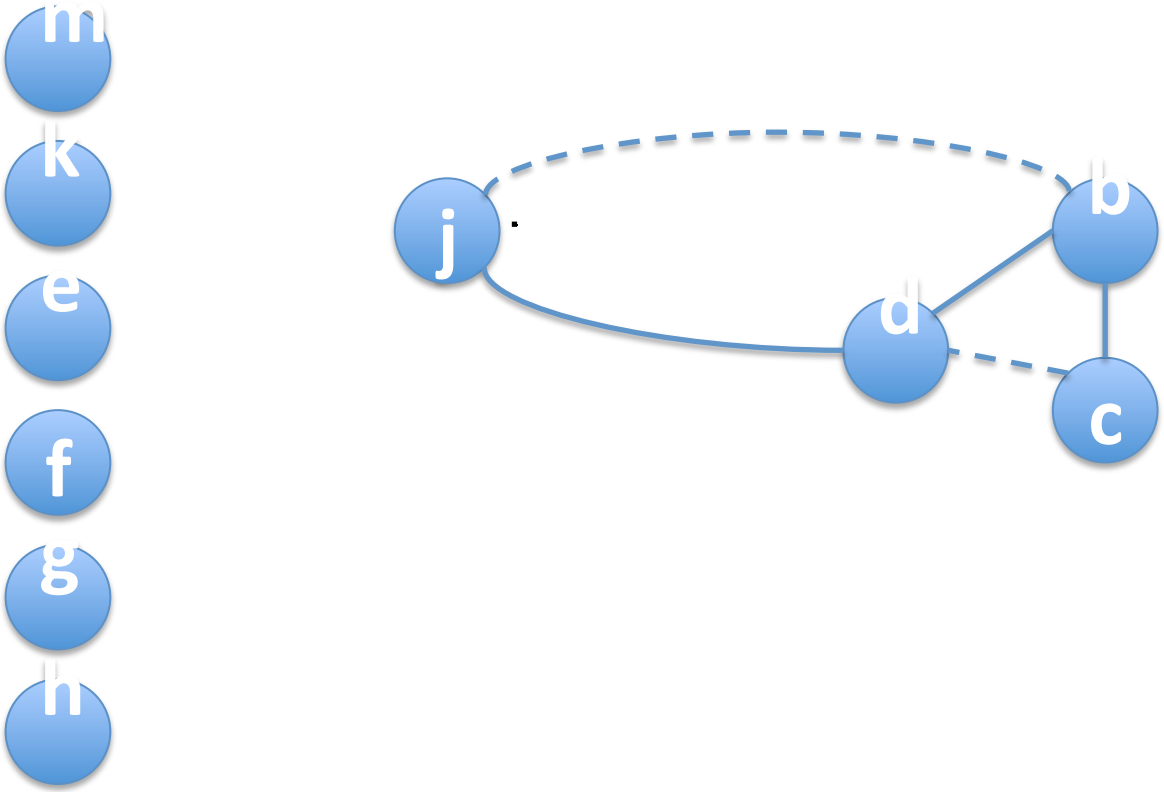
Retomando o Exemplo

- Agora podemos remover: m



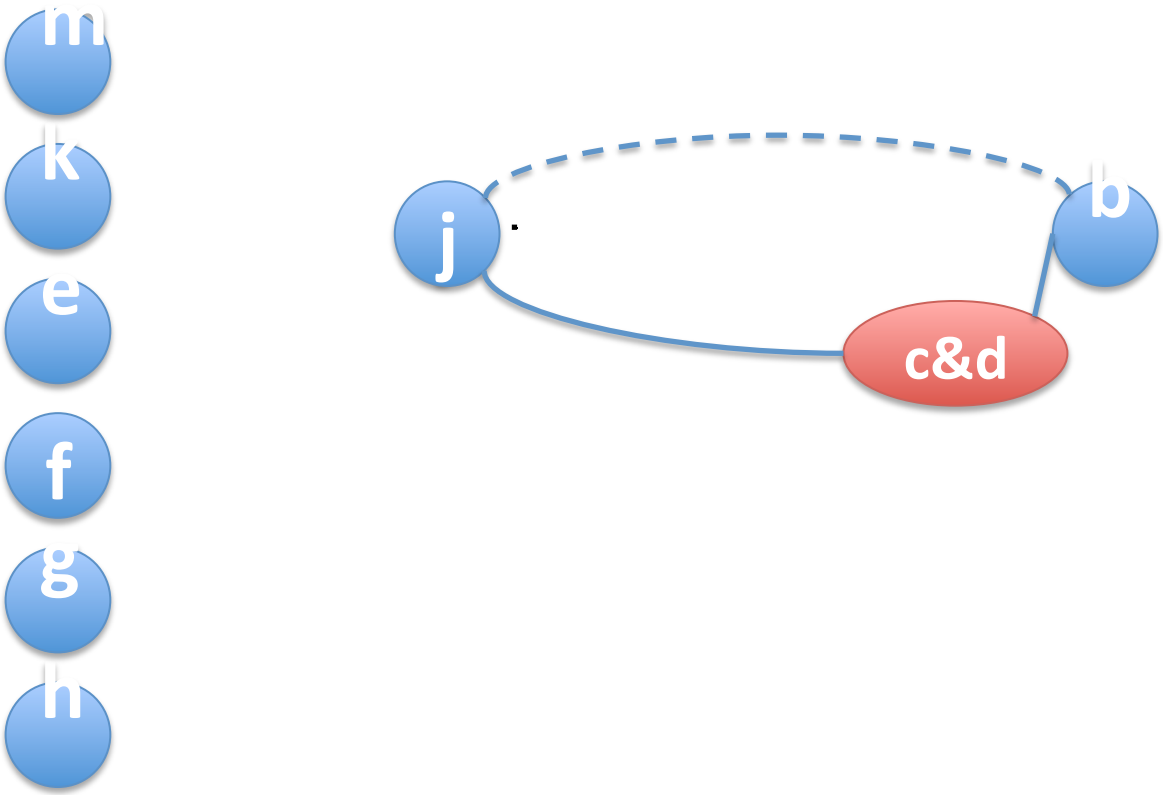
Retomando o Exemplo

- Sobraram apenas nós *move-related.* Aplicamos *coalescing:*



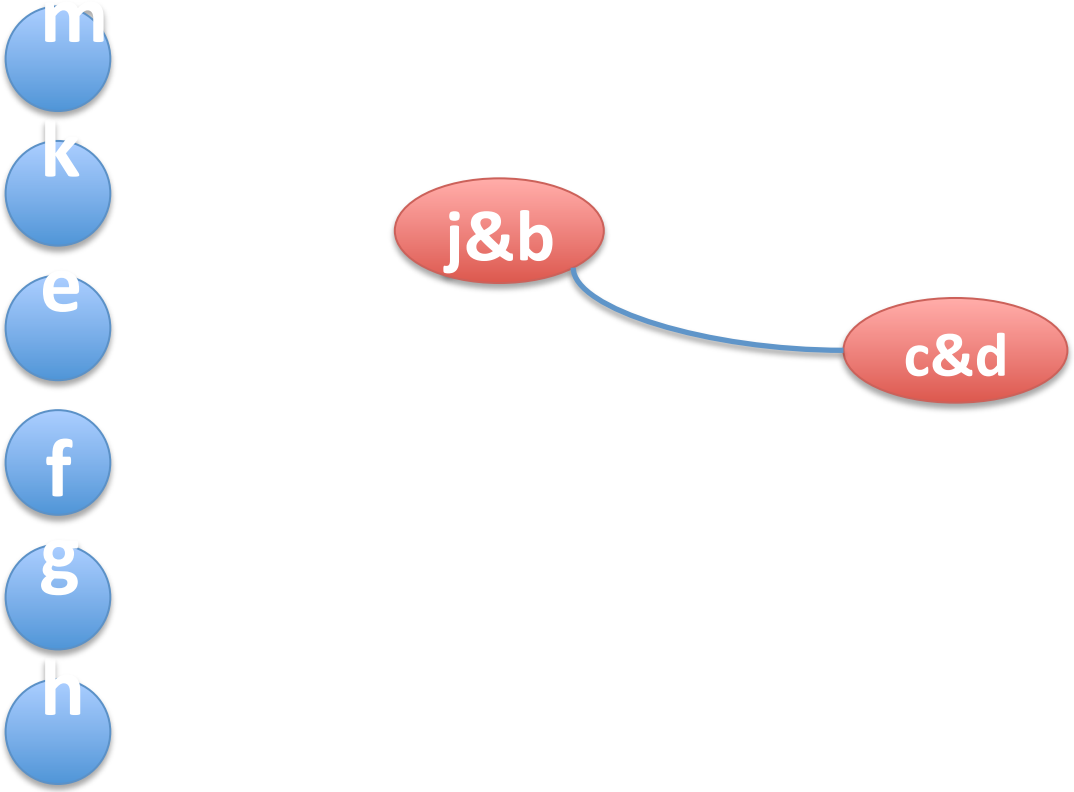
Retomando o Exemplo

- Sobraram apenas nós *move-related.* Aplicamos *coalescing:*



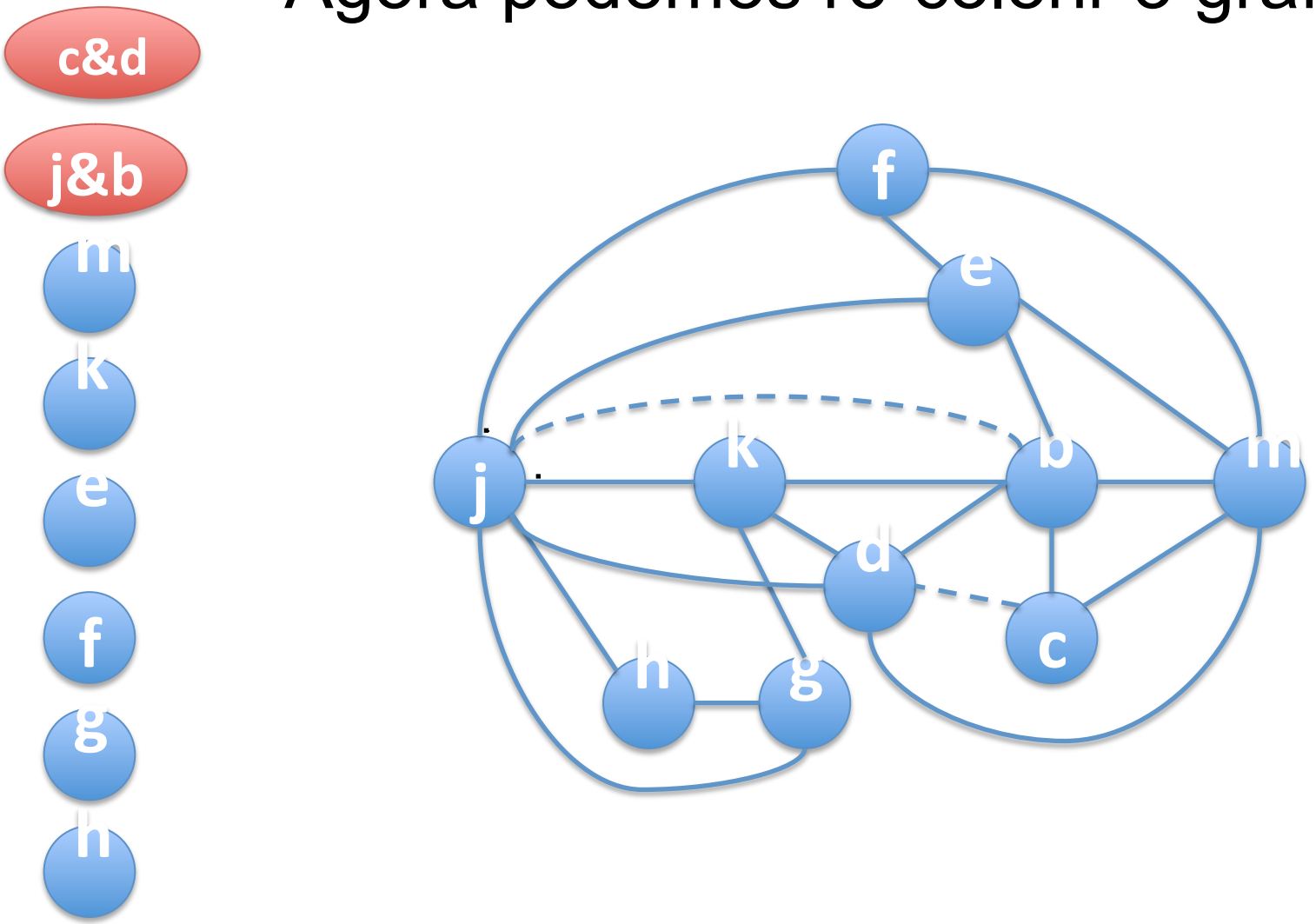
Retomando o Exemplo

- Não há mais oportunidade para coalescing. Voltamos ao *simplify*



Retomando o Exemplo

- Agora podemos re-colorir o grafo.



Spilling com *Coalescing*

- Solução simples:
  - Descartar todos os *coalescing* feitos quando recomeçar o Build.
- Mais eficiente:
  - Conservar os *coalescing* feitos antes do primeiro *potencial spill.*
  - Descarta os subsequentes.

Coalescing de *Spills*

- Muitos registradores => poucos *spills*
- Poucos registradores => vários *spills*
  - Aumenta o tamanho dos registros de ativação (AR)
  - Ex. Pentium: 6 registradores
- Transformações/Otimizações
  - Podem gerar mais temporários
- O frame da função pode ficar grande

Coalescing de *Spills*

- Instruções MOVE envolvendo valores que sofreram *spill:*
  - a ← b implica em:
    - t ← M[b]
    - M[a] ← t
  - Caro e ainda cria mais um valor temporário
- Muitos dos valores que sofrem spill não estão vivos simultaneamente.
- Podemos usar a mesma técnica que para registradores!

Coalescing de *Spills*

- Coloração com *coalescing* para os *spills*

1. Use o *liveness* para construir um IG para os *spills*
2. Enquanto houver *spills* sem interferência e com MOVE
   - Una esses nós (*Coalescing*)
3. Use *simplify* e *select* para colorir o grafo

Coalescing de *Spills*

3. Use *simplify* e *select* para colorir o grafo
   - **Não existe** *spill* nesta coloração
   - *Simplify* vai retirando o nó de menor grau até o fim
   - *Select* vai escolhendo a menor cor possível
     - Sem limite, pois não temos limite para o tamanho do frame
4. As cores correspondem a posições do registro de ativação (*stack frame*) da função.

- Fazer **antes** da reescrita do código

Pré-coloração

- Alguns nós do IG podem ser pré-coloridos
  - Temporários associados ao FP, SP, registradores de passagem de argumentos.
  - Permanentemente associados aos registradores físicos.
  - Cores pré-definidas e únicas.
  - Podem ser reaproveitados no *select* e *coalesce*
    - Desde que não interfiram com o outro valor
    - Ex. Um registrador de passagem de parâmetro pode servir como temporário no corpo da função

Pré-coloração

- Podem ser unidos no *coalescing* com outros nós não pré-coloridos.
- *Simplify* os trata como tendo grau “infinito”.
  - Não devem ir para a pilha.
  - Não devem sofrer *spill*.
- O algoritmo executa *simplify, select* e *spill* até sobrarem somente nós pré-coloridos.

Pré-coloração

- Podem ser copiados para temporários
  - Suponha que r7 seja um *callee-save register*

```
enter: def(r7)

       ...

exit:  use(r7)
```

```
enter: def(r7)
       t231 := r7
       ...
       r7 := t231
exit:  use(r7)
```

Exemplo

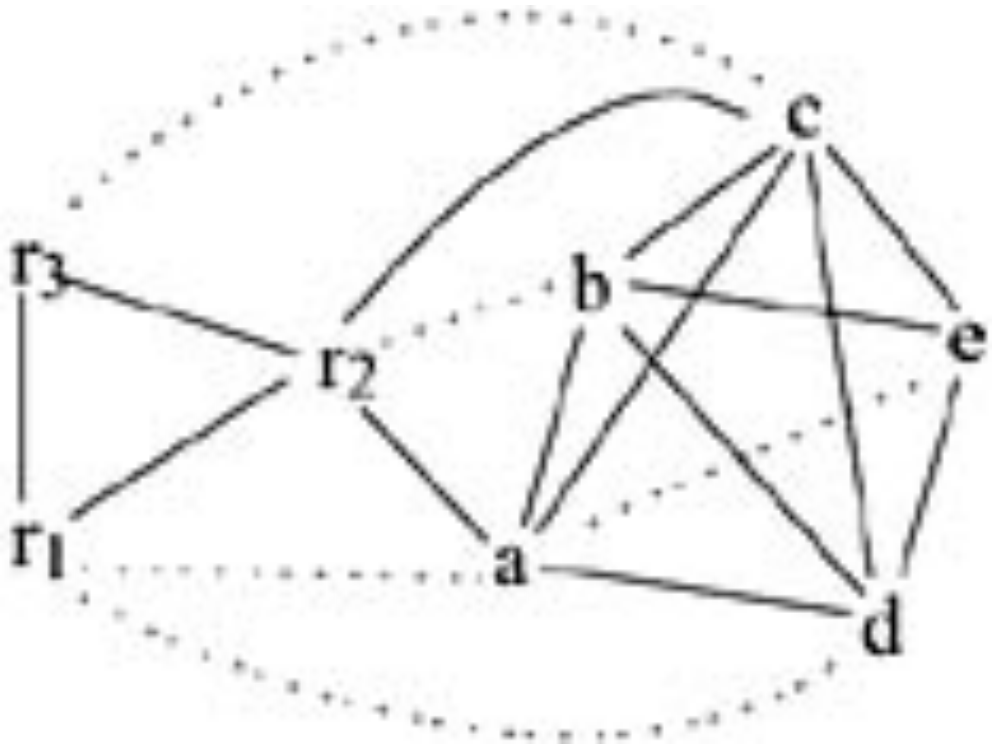
- Três registradores:
  - R1 e r2 *caller-save*
  - R3 *callee-save*

```
int f(int a, int b)
{
  int d = 0;
  int e = a;
  do { d = d + b;
       e = e – 1;
  } while (e>0);
  return d;
}
```

```
enter:  c := r3
        a := r1
        b := r2
        d := 0
        e := a
loop:   d := d + b
        e := e – 1
        if e > 0 goto loop
        r1 := d
        r3 := c
        return
        ;(r1, r3 live out)
```

Exemplo

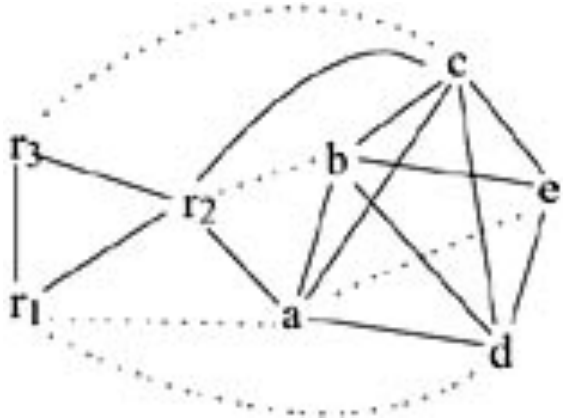
- IG para o programa anterior
  - K = 3
  - Tem oportunidades para *simplify* e *spill*?



Exemplo

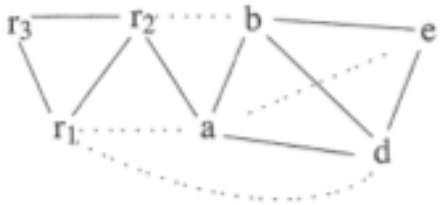
- Veja cálculo de prioridade para o *spill* na tabela do livro

Node	Uses+Defs outside loop		Uses+Defs within loop		Degree		Spill priority
a	(2	+ 10 ×	0)	/	4	=	0.50
b	(1	+ 10 ×	1)	/	4	=	2.75
c	(2	+ 10 ×	0)	/	6	=	0.33
d	(2	+ 10 ×	2)	/	4	=	5.50
e	(1	+ 10 ×	3)	/	3	=	10.33

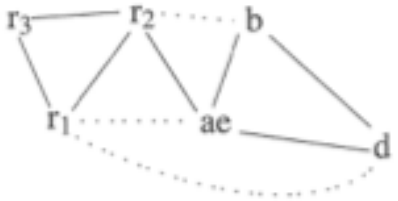


Exemplo

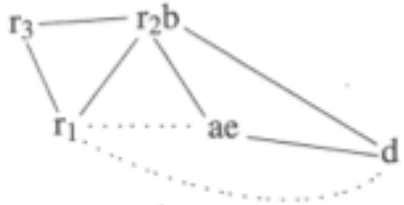
Node c has the lowest priority – it interferes with many other temporaries but is rarely used – so it should be spilled first. Spilling c, we obtain the graph at right.



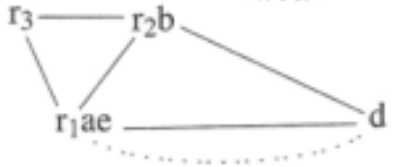
2. We can now coalesce a and e, since the resulting node will be adjacent to fewer than K significant-degree nodes (after coalescing, node d will be low-degree, though it is significant-degree right now). No other *simplify* or *coalesce* is possible now.



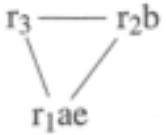
3. Now we could coalesce $ae\&r_1$ or coalesce $b\&r_2$. Let us do the latter.



4. We can now coalesce either $ae\&r_1$ or coalesce $d\&r_1$. Let us do the former.



5. We cannot now coalesce $r_1ae\&d$ because the move is *constrained:* The nodes r_1ae and d interfere. We must *simplify* d.



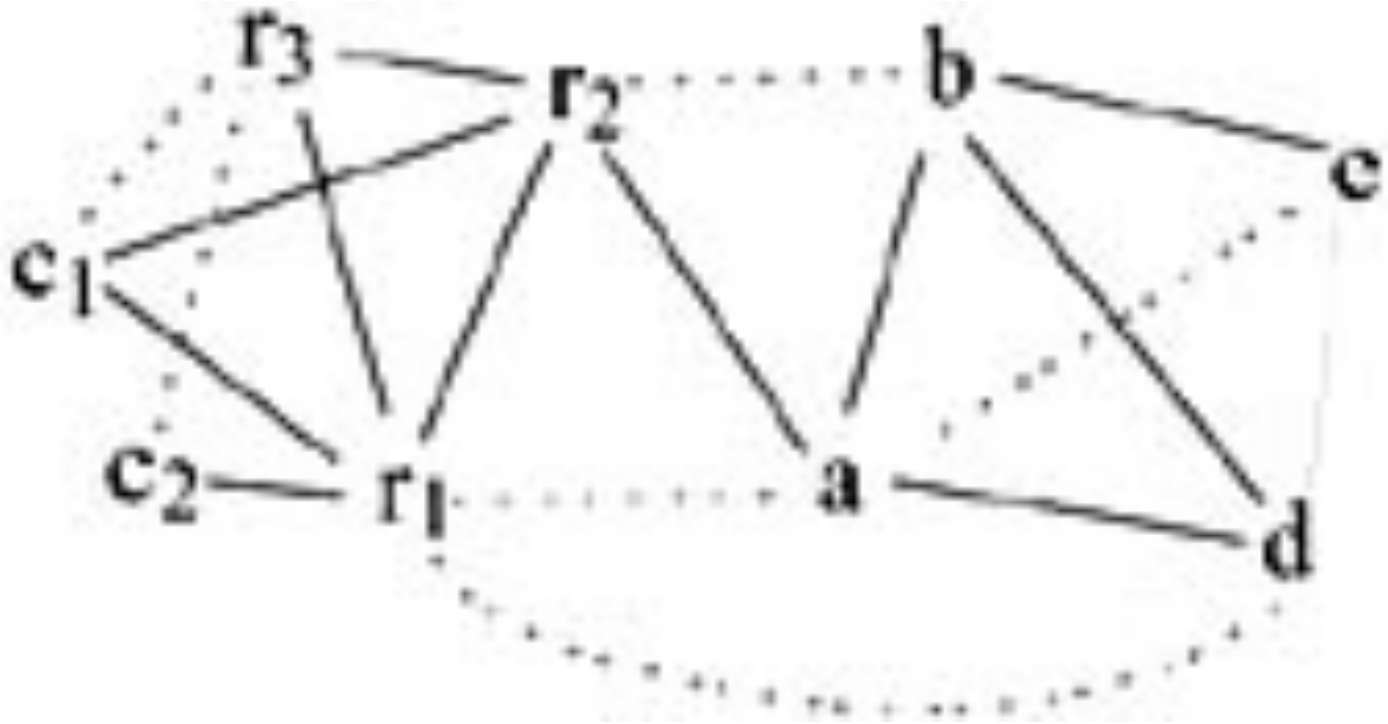
Exemplo

- Código após reescrita gerada pelo *spill* de c

```
enter:  c1 := r3
        M[Cloc] := c1
        a := r1
        b := r2
        d := 0
        e := a
loop:   d := d + b
        e := e – 1
        if e > 0 goto loop
        r1 := d
        c2 := M[Cloc]
        r3 := c2
        return
        ;(r1, r3 live out)
```

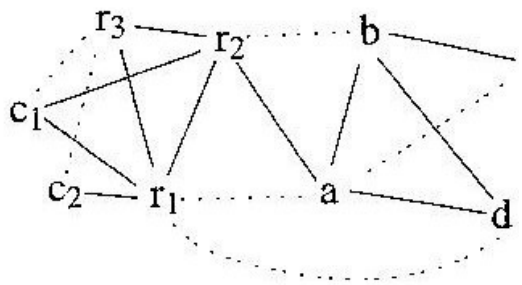
Exemplo

- Novo IG

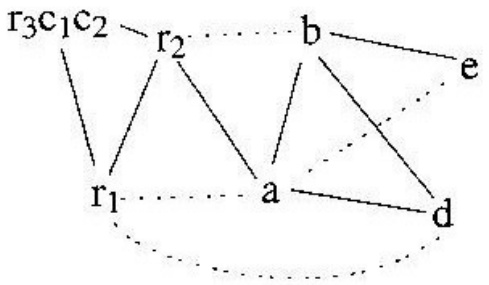


Exemplo

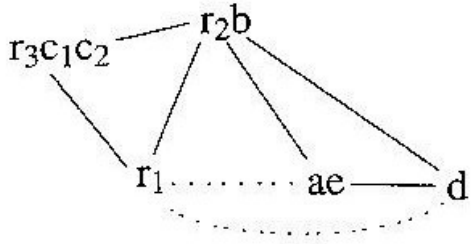
8. Now we build a new interference graph:



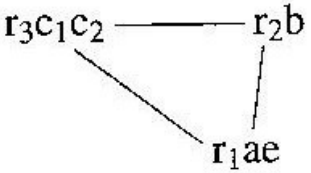
9. Graph-coloring proceeds as follows. We can immediately coalesce $c_1 \& r_3$ and then $c_2 \& r_3$.



10. Then, as before, we can coalesce $a \& e$ and then $b \& r_2$.



11. As before, we can coalesce $ae \& r_1$ and then simplify d.



Exemplo

- Código alocado

Node	Color
a	r_1
b	r_2
c	r_3
d	r_3
e	r_1

```
enter:  r3 := r3
        M[Cloc] := r3
        r1 := r1
        r2 := r2
        r3 := 0
        r1 := r1
loop:   r3 := r3 + r2
        r1 := r1 – 1
        if r1 > 0 goto loop
        r1 := r3
        r3 := M[Cloc]
        r3 := r3
        return
```

Exemplo

- Código com MOVES eliminados

```
enter: r3 := r3
       M[Cloc] := r3
       r1 := r1
       r2 := r2
       r3 := 0
       r1 := r1
loop:  r3 := r3 + r2
       r1 := r1 – 1
       if r1 > 0 goto loop
       r1 := r3
       r3 := M[Cloc]
       r3 := r3
       return
```

```
enter: M[Cloc] := r3
       r3 := 0
loop:  r3 := r3 + r2
       r1 := r1 – 1
       if r1 > 0 goto loop
       r1 := r3
       r3 := M[Cloc]
       return
```