Testes baseados no fluxo de dados
Grafos interprocedimentais

Criado: abril/2003

Últ. atualização: mar/2013

Tópicos

- Anomalias de fluxo de dados
- Algumas definições
- Grafo de fluxo de dados
- Critérios de cobertura
- Análise de mutantes

Referências

R.Binder. *Testing Object-Oriented Systems. Models, Patterns and Tools.* Addison Wesley, 2000, cap. 10.3.

P.Jalote. “An Integrated Approach to Sw Engineering”, 2ª ed., 1997, cap. 9.3.2.

P. Amman e J.Offutt. *Introduction to Software Testing*. Cambridge University Press, 2008, c. 2.

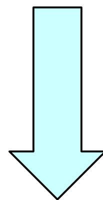
Introdução

- Testes baseados na implementação ou testes caixa-branca:
  - são baseados no código
  - visam exercitar estruturas de controle (instruções) e de dados de um programa

- Modelos de teste:
  - grafo de fluxo de controle
  - **grafo de fluxo de dados**

Princípio

- Programa = seqüência de ações realizadas sobre variáveis



fluxo de controle

+

informações sobre onde variáveis são definidas
e onde essas definições são usadas

Qual o modelo de falhas ?

- Anomalias de fluxo de dados:
  - uso de variável não inicializada
  - atribuição valor a uma variável mais de uma vez sem que tenha havido uma referência a essa variável entre essas atribuições
  - liberação ou reinicialização de uma variável antes que ela tenha sido criada ou inicializada
  - liberação ou reinicialização de uma variável antes que ela tenha sido usada
  - atribuir novo valor a um ponteiro sem que variável tenha sido liberada

Algumas definições

- Uma ocorrência de uma variável em um programa pode ser:
  - **def**, representando a definição de uma variável, em que um valor é atribuído a essa variável ex.: read X; A := 1;
  - **uso**, representando a referência uma variável, caso em que um valor já deve ter sido atribuído a essa variável.
    Um uso pode ser:
    - **p-uso**, representando a ocorrência da variável em predicados (expressões lógicas ou relacionais) ex.: X > Y or not Z
    - **c-uso**, representando a ocorrência da variável em expressões aritméticas ex.: X + 1

Algumas definições

- **c-uso global** quando não existe **def** da variável no bloco em que ocorre c-uso
- **caminho livre de definição (c.l.d.)** com relação a uma variável x: caminho entre blocos A e B em que não há def de x entre A e B
- **def global**: quando **def** de uma variável x em um bloco A é usada em um bloco B (ou em um predicado) sem que haja redefinição de x entre os blocos A e B   ou

  existe c.l.d.(x) entre A e B

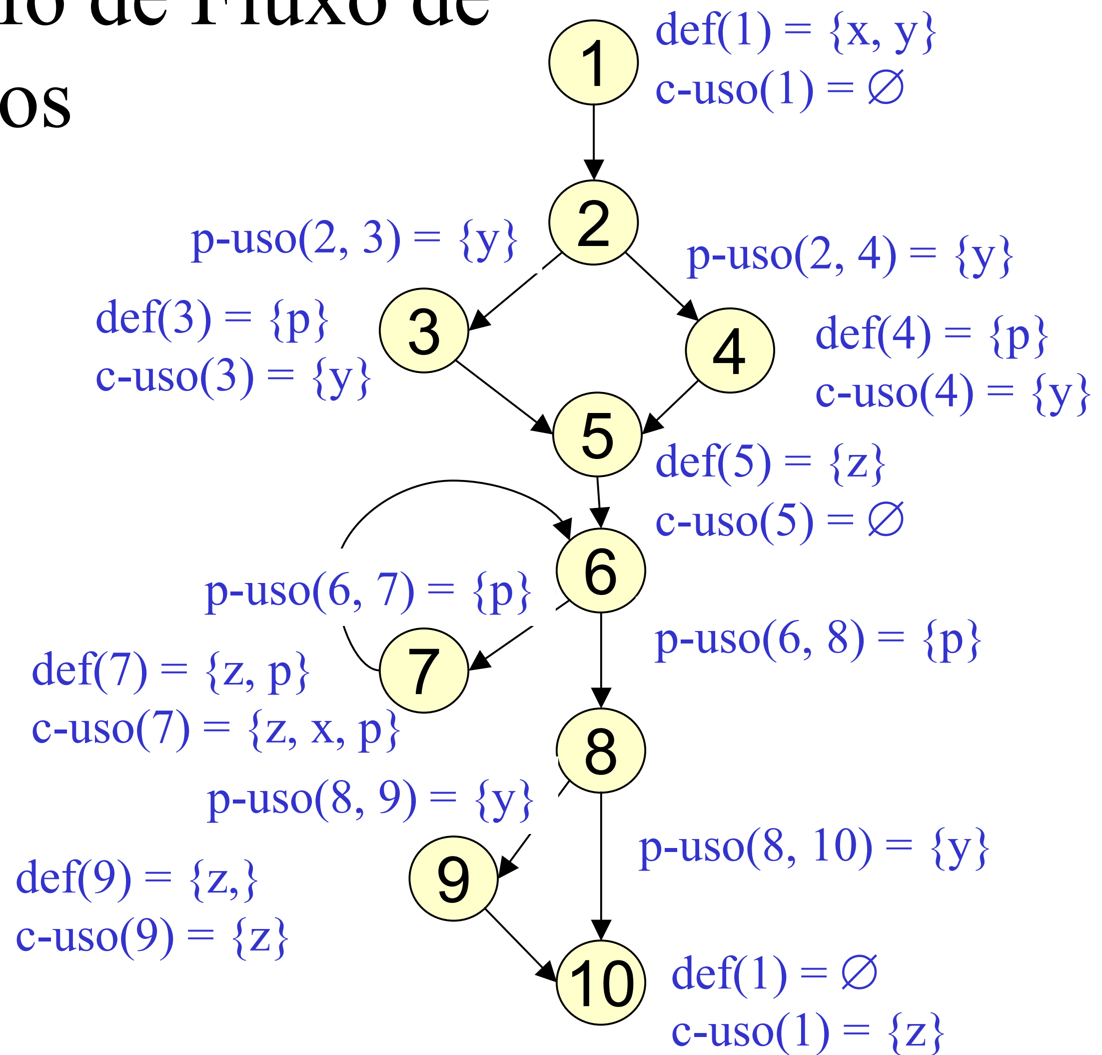
Grafo de fluxo de dados

- Grafo de fluxo de dados (ou grafo def / uso) é o grafo de fluxo de controle ao qual associa-se conjuntos de variáveis a seus nós e arcos
- Conjuntos:
  - **def (i)** = conjunto de variáveis com definição global no nó i
  - **c-uso(i)** = conjunto de variáveis com c-uso global no nó i
  - **p-uso (i, j)** = conjunto de variáveis com p-uso na aresta (i, j)

Exemplo de Grafo de Fluxo de Dados

Cálculo de x^y

```
1.  read x, y;
2.  if y < 0
3.    then p := 0 - y
4.    else p := y;
5.  z := 1.0;
6.  while p ≠ 0 do
7.  begin
      z := z * x; p := p - 1;
    end;
8.  if y < 0
9.   then z := 1 / z;
10. write z;
    end;
```



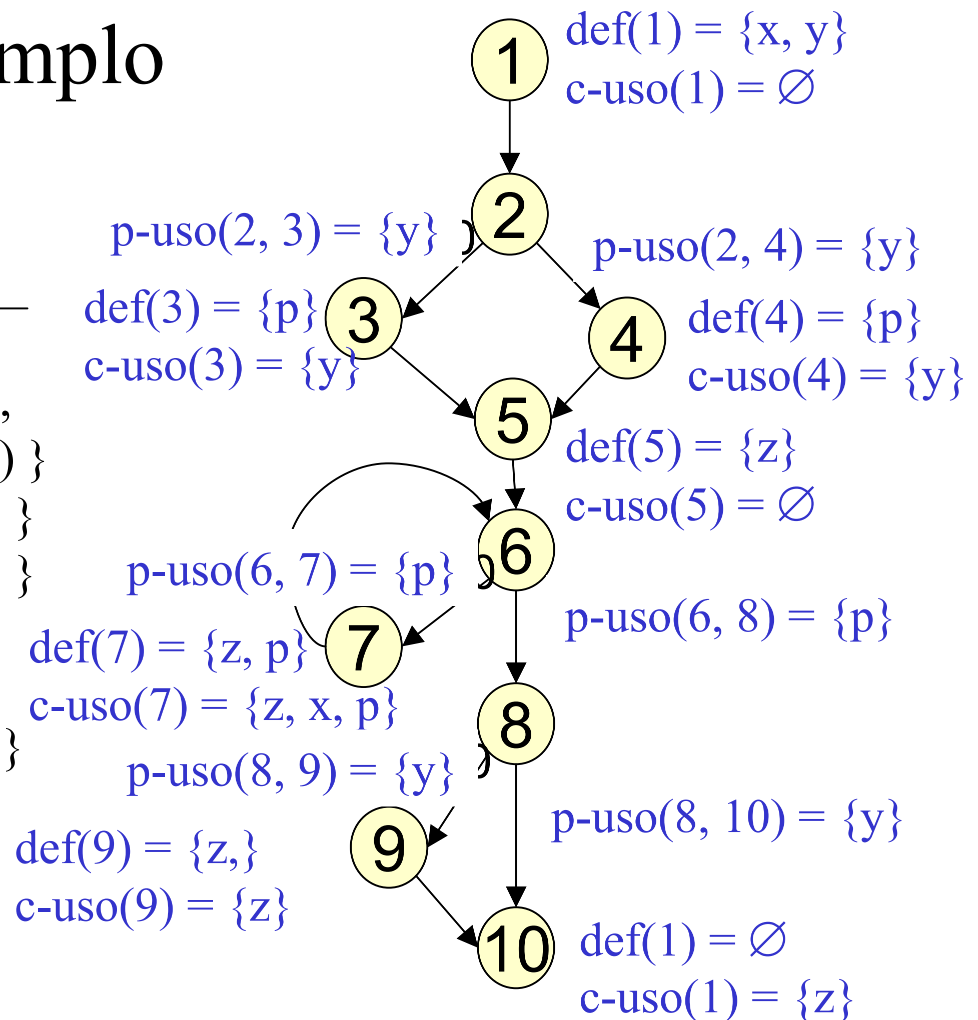
Mais conjuntos

- Suponha que x é uma variável, i é um nó tal que $x \in def(i)$:
  - **dcu (x, i)** = conjunto de nós $j \neq i$ tal que $x \in c\text{-}uso(j)$ e $\exists$ cl.d. (x) de i a j
  - **dpu (x, i)** = conjunto de arcos (j, k) onde $j \neq i, k \neq i$ tal que $x \in p\text{-}uso(j, k)$ e $\exists$ cl.d. (x) de i a (j, k)

Com base nesses conjuntos foram definidos critérios de teste def-uso também conhecidos como família de critérios de Rapps e Weyuker (1985)

Exemplo

(nó, variável)	dcu	dpu
(1, x)	{ 7}	∅
(1, y)	{3, 4}	{ (2, 3), (2, 4), (8, 9), (8, 10) }
(3, p)	{ 7 }	{ (6, 7), (6, 8) }
(4, p)	{ 7 }	{ (6, 7), (6, 8) }
(5, z)	{ 7, 9, 10 }	∅
(7, z)	{7, 9, 10}	∅
(7, p)	{ 7 }	{ (6, 7), (6, 8) }
(9, z)	{9, 10}	∅



Critérios def-uso

- Objetivos:
  - exercitar caminhos ligando definições globais a usos globais de variáveis do programa
- Tipos:
  - todas as definições
  - todos os p-usos
  - todos os p-usos e alguns c-usos
  - todos os c-usos e alguns p-usos
  - todos os usos

Critérios def-uso

- Sejam
  - G - grafo def/uso de um programa,
  - T: conjunto de testes
  - p(T): conjunto de caminhos de teste em G executados por T
- p(T) satisfaz ao critério **todas as definições** se
  - ∀ nó i ∈ G e ∀ variável x ∈ def (i), p(T) inclui um c.l.d.(x) para algum nó j ∈ dcu(i) ou para alguma aresta (j, k) ∈ dpu(i)
    ou seja,

    para todas as definições de variáveis deve ser exercitado um caminho para um de seus usos

Exemplo: critério todas as definições

(nó, variável)	dcu	dpu
(1, x)	{ 7}	∅
(1, y)	{3, 4}	{ (2, 3), (2, 4), (8, 9), (8, 10) }
(3, p)	{ 7 }	{ (6, 7), (6, 8) }
(4, p)	{ 7 }	{ (6, 7), (6, 8) }
(5, z)	{ 7, 9, 10 }	∅
(7, z)	{7, 9, 10}	∅
(7, p)	{ 7 }	{ (6, 7), (6, 8) }
(9, z)	{9, 10}	∅

Caminhos:
1-2-3-5-6-7-6-8-9-10
1-2-4-5-6-7-6-8-10

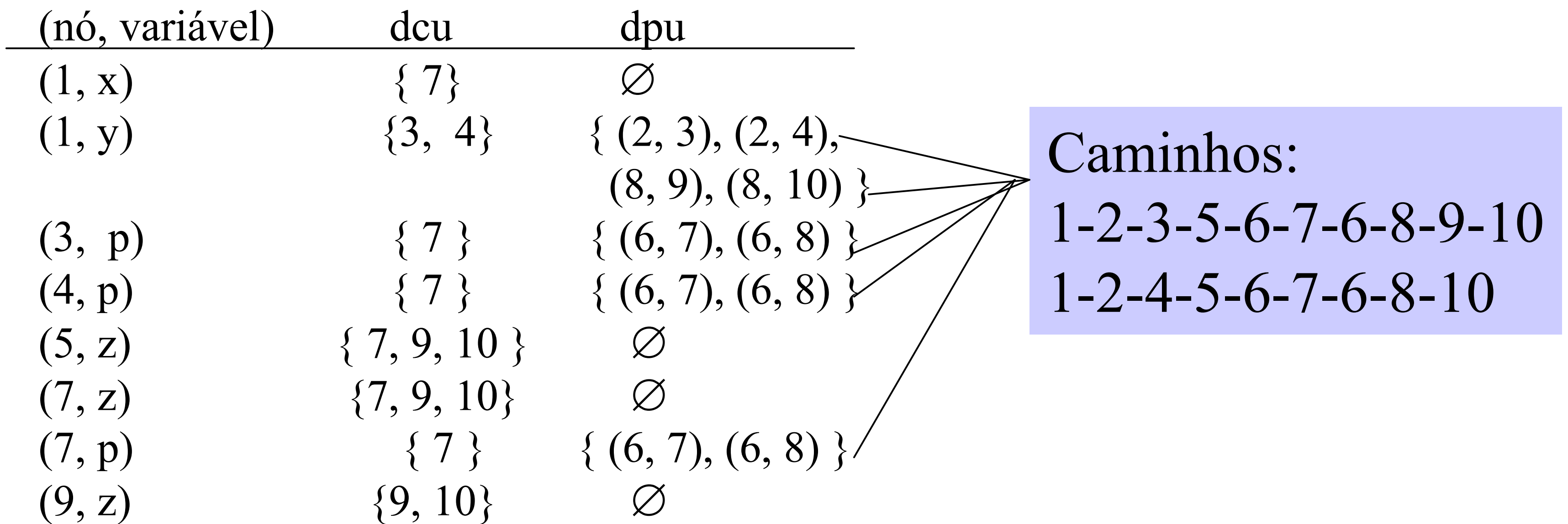
Critérios def-uso

- p(T) satisfaz ao critério **todos os p-usos** se
  - $\forall$ nó i $\in$ G e $\forall$ variável x $\in$ def (i), p(T) inclui um c.l.d.(x) para alguma aresta (j, k) $\in$ dpu(i)
    ou seja,

    para todas as definições de variáveis deve ser exercitado um caminho para todos os seus p-usos

Exemplo: critério todos os p-usos

(nó, variável)	dcu	dpu
(1, x)	{ 7}	∅
(1, y)	{3, 4}	{ (2, 3), (2, 4), (8, 9), (8, 10) }
(3, p)	{ 7 }	{ (6, 7), (6, 8) }
(4, p)	{ 7 }	{ (6, 7), (6, 8) }
(5, z)	{ 7, 9, 10 }	∅
(7, z)	{7, 9, 10}	∅
(7, p)	{ 7 }	{ (6, 7), (6, 8) }
(9, z)	{9, 10}	∅



Critérios def-uso

- p(T) satisfaz ao critério **todos os p-usos e alguns c-usos** se
  - $\forall$ nó i $\in$ G e $\forall$ variável x $\in$ def (i), p(T) inclui um c.l.d.(x) para todas as arestas (j, k) $\in$ dpu(i) e para algum nó j $\in$ dcu(i)
    ou seja,

    para todas as definições de variáveis deve ser exercitado um caminho para todos os seus p-usos e alguns c-usos
- p(T) satisfaz ao critério **todos os c-usos e alguns p-usos** se
  - $\forall$ nó i $\in$ G e $\forall$ variável x $\in$ def (i), p(T) inclui um c.l.d.(x) para todos os nós j $\in$ dcu(i) para algumas arestas (j, k) $\in$ dpu(i)
    ou seja,

    para todas as definições de variáveis deve ser exercitado um caminho para todos os seus c-usos e para alguns p-usos

Critérios def-uso

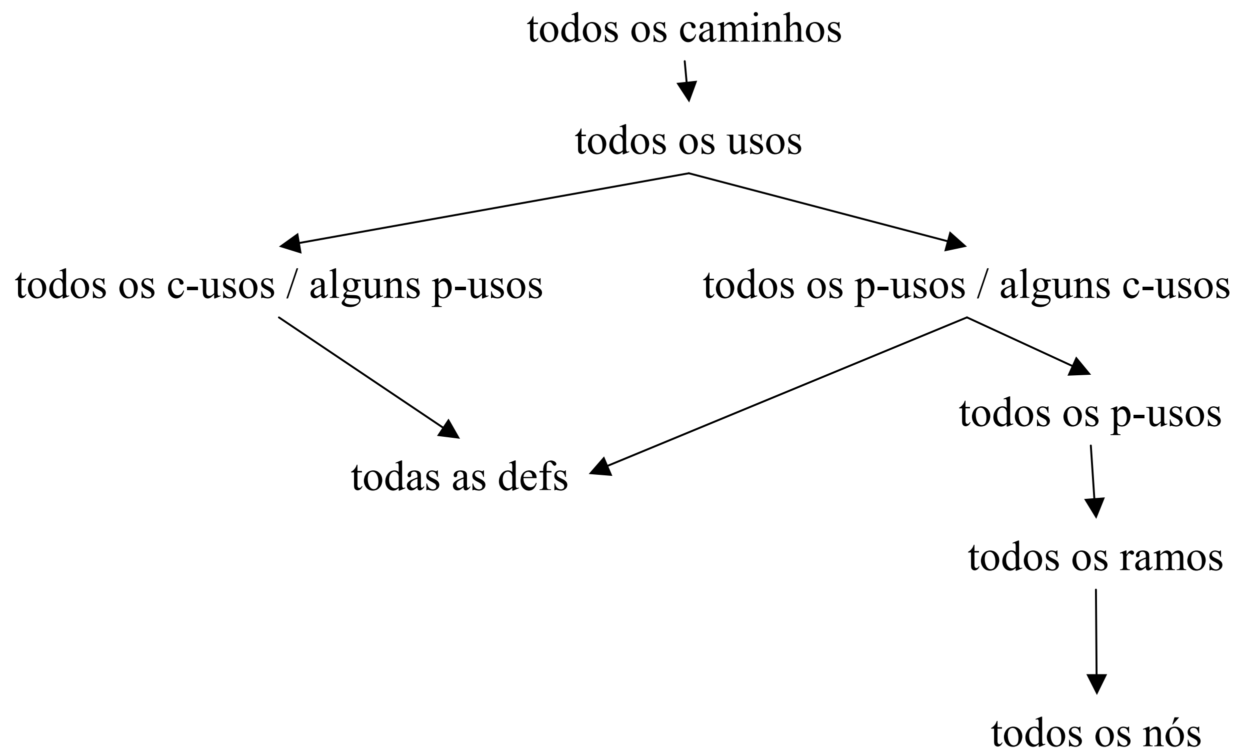
- p(T) satisfaz ao critério **todos os usos** se
  - $\forall$ nó i $\in$ G e $\forall$ variável x $\in$ def (i), p(T) inclui um c.l.d.(x) para todos os nós j $\in$ dcu(i) para todas as arestas (j, k) $\in$ dpu(i)
    ou seja,

    para todas as definições de variáveis deve ser exercitado um caminho para todos os seus c-usos e para todos os seus p-usos

Relação de ordem entre os critérios

- Um critério C1 inclui estritamente um critério C2 se:
  - todo conjunto de teste que cobre C1 cobre também C2
  - existe(m) conjunto(s) de teste que cobre(m) C2 mas não cobre(m) C1
- A relação de inclusão não implica que um critério seja melhor do que outro. Só indica que, se a estratégia usada para gerar casos de testes para C1 e C2 é a mesma os conjuntos de teste que satisfazem C1 serão melhores que o conjunto de testes que satisfazem C2

Relação de inclusão entre os critérios caixa branca



Exercício 1 - Rotina para busca binária

```
Entradas: tabela, item, chave
Saídas: achou, onde
comeco := 1;
fim := Tamanho_tabela;
achou := falso;
while comeco ≤ fim and not achou do
    meio := (comeco + fim) / 2;
    if chave > tabela [meio]
        then comeco := meio + 1
        else if chave = tabela [meio]
            then achou := verdade;
                    onde := meio
            else fim := meio - 1
end while;
```

Exercício 1 - Rotina para busca binária

Entradas: tabela, item, chave
Saídas: achou, onde

```
1. comeco := 1;
   fim := Tamanho_tabela;
   achou := falso;
2. while comeco ≤ fim and not achou do
3.    meio := (comeco + fim) / 2;
4.    if chave > tabela [meio]
5.       then comeco := meio + 1
6.       else if chave = tabela [meio]
7.          then achou := verdade;
                 onde := meio
8.          else fim := meio - 1
9. end while;
```

Exercício 2

```
Entradas: x1, x2, x3: inteiros;
              x4: char
Saídas: w, x1
if x1 > x2
   then w := 100
   else w := 10;
while x1 > x3 do
   x3 := x3 * w
 end while;
case x4
      “A” - “J”: w := 10
      “K” - “T”: w := 20
      “U” - “Z”: w := 30
end case;
```

```
if x2 > x1 and x2 < x3 and x4 = “S”
   then w := 10
   else w := 20;
if x1 < w or x2 < w
   then write w
   else write x1;
```

Testes baseados em grafos (mais de uma unidade)

Tópicos

- Fluxo de controle entre procedimentos
- Grafo de chamadas
- Fluxo de controle com tratamento de exceções
- Fluxo de dados entre procedimentos

Referências

P. Amman e J.Offutt. *Introduction to Software Testing*. Cambridge University Press, 2008, c. 2.4.

G.Rothermel, M.J.Harrold, J.Dedhia. “Regression Test Selection for C++ Software”. Technical Report 99-60-01, Computer Science Dept., Oregon State University (OSU), Jan/1999.

M.J.Harrold, M.L.Soffa. “Interprocedual data flow testing”. Proceedings of the ACM SIGSOFT '89 third symposium on Software testing, analysis, and verification, Key West, Florida, United States, 1989, pp158 - 167 .

T.Reps, S.Horwitz, M.Sagiv. “Precise Interprocedural Dataflow Analysis via Graph Reachability” , Conference Record of the Twenty-Second ACM Symposium on Principles of Programming Languages, (San Francisco, CA, Jan. 23-25, 1995), pp. 49-61.

S.Sinha, M.J.Harrold. “Control-Flow Analysis of Programs with Exception-Handling Constructs”. Technical Report OSU-CISRC-7/98-TR25, Jul/1998.

Como testar com base na estrutura do projeto?

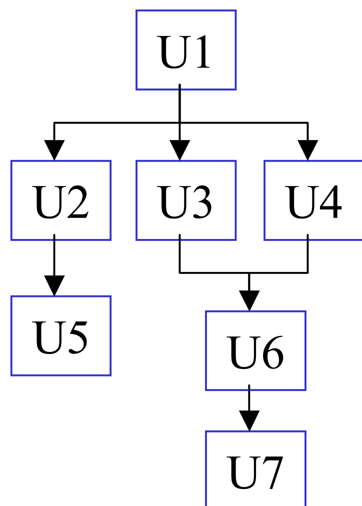
- Projeto detalhado:
  - Possibilidade de se ter diversos procedimentos
- Grafos:
  - Devem representar relacionamentos entre os procedimentos
  - Duas possibilidades:
    - Grafo de fluxo de controle
    - Grafo de fluxo de dados

Representação do fluxo de controle

- Grafo de chamadas
- Grafo interprocedimental

Grafos de chamadas

- Grafos de chamadas (*call graph*):
  - Usados para representar relações entre procedimentos em projetos estruturados (também chamados de Diagrama de Estrutura do Sistema)
  - Nós = 1 unidade (ex.: procedimento ou método)
  - Arestas = chamadas às unidades

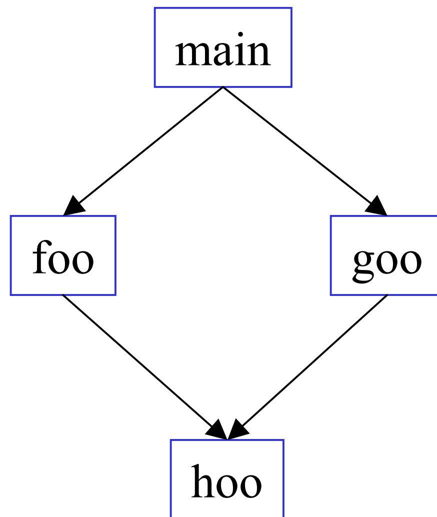


Cobertura de Nós: chame cada unidade pelo menos 1 vez (cobertura de funções)

Cobertura de Arestas: execute cada chamada pelo menos 1 vez (cobertura de chamadas)

Grafo de chamadas - exemplo

```
void foo(int b)
    { hoo(b); }
void goo(int c)
    { hoo(c); }
main()
{
    int x, y;
    foo(x);
    goo(y);
}
```



Requisitos de testes

Cobertura de nós:
{main( ), foo( ), goo( ), hoo( )}

Cobertura de arestas:
{main( )-foo( ), main( )-goo( ), foo( )-hoo( ), goo( )-hoo( )}

Grafo de chamadas - considerações

☺ Vantagens:
- Precisão na representação do projeto

☹ Desvantagens:
- Crescimento exponencial ➔ precisa de uma forma para tratar recursão
- Em OO: nem todos os métodos chamam outros → não dá para construir um grafo de chamadas

Grafo de fluxo de controle entre procedimentos

- Um Grafo de Fluxo de Controle (GFC) representa o fluxo de um único procedimento (ou método)
- Grafo de Fluxo de Controle Interprocedimental (GFCI) :
  - Conjunto de GFCs interligados
  - Cada GFC representa o fluxo de controle de um procedimento
  - Um GFC é aumentado com:
    - Com **nós de entrada e saída**
    - A cada chamada de procedimento, acrescenta-se ao GFC **nós de chamada e retorno**
  - Arestas interligam os GFCs nos pontos de chamada e retorno: **arestas de chamada e retorno**

Fluxo de controle entre procedimentos

1. declare g: **integer**

2. program *main*

3. begin

4. declare x: **integer**

5. read(x)

6. call P (x)

7. end

8. procedure P (**value** a : **integer**)

9. begin

10. if ($a > 0$) **then**

11. read(g)

12. $a := a - g$

13. call P (a)

14. print(a, g)

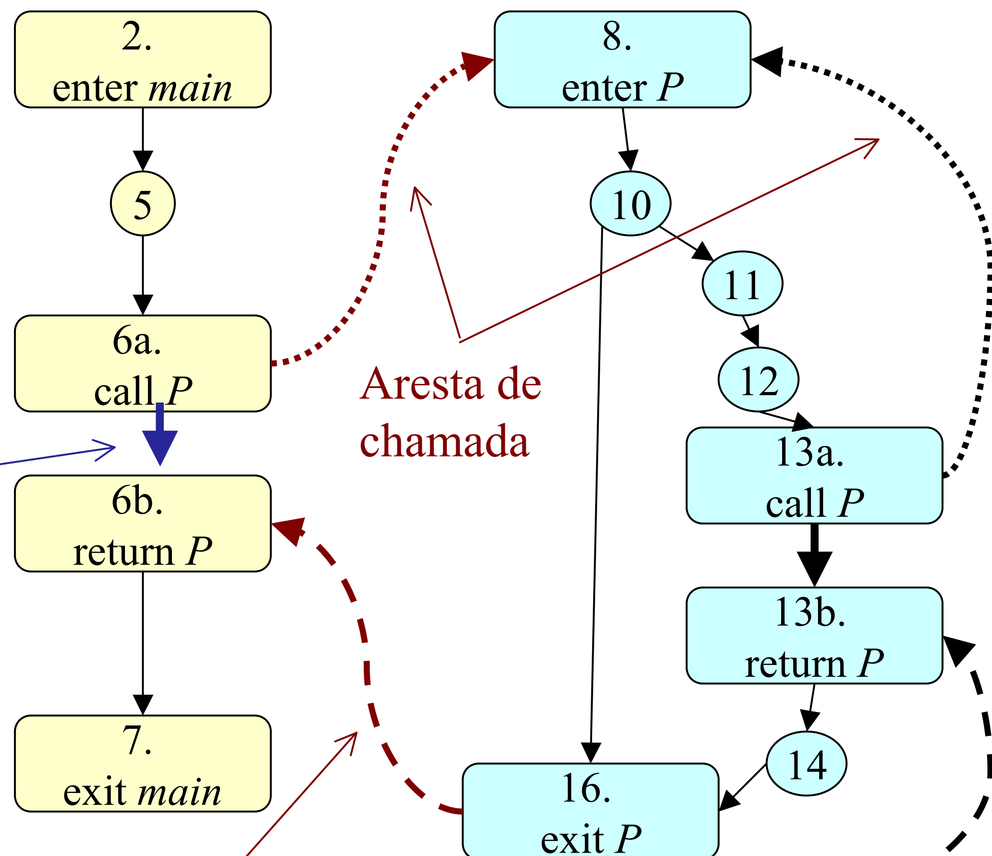
15. fi

16. end

[Reps et al. 1995]

Fluxo de controle entre procedimentos

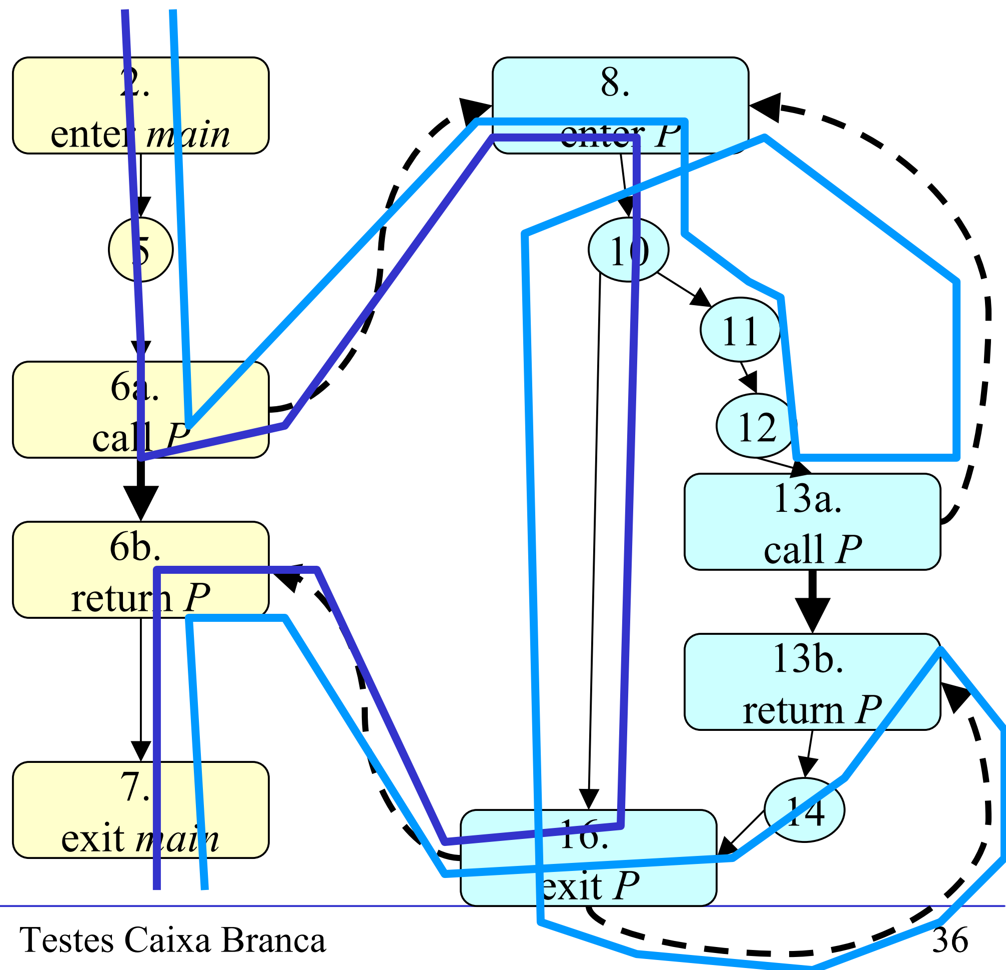
1. **declare** g: **integer**
2. **program** *main*
3. **begin**
4. **declare** x: **integer**
5. **read**(x)
6. **call** P (x)
7. **end**
8. **procedure** P (**value** a : **integer**)
9. **begin**
10. **if** ($a > 0$) **then**
11. **read**(g)
12. $a := a - g$
13. **call** P (a)
14. **print**(a, g)
15. **fi**
16. **end**



[Reps et al. 1995]

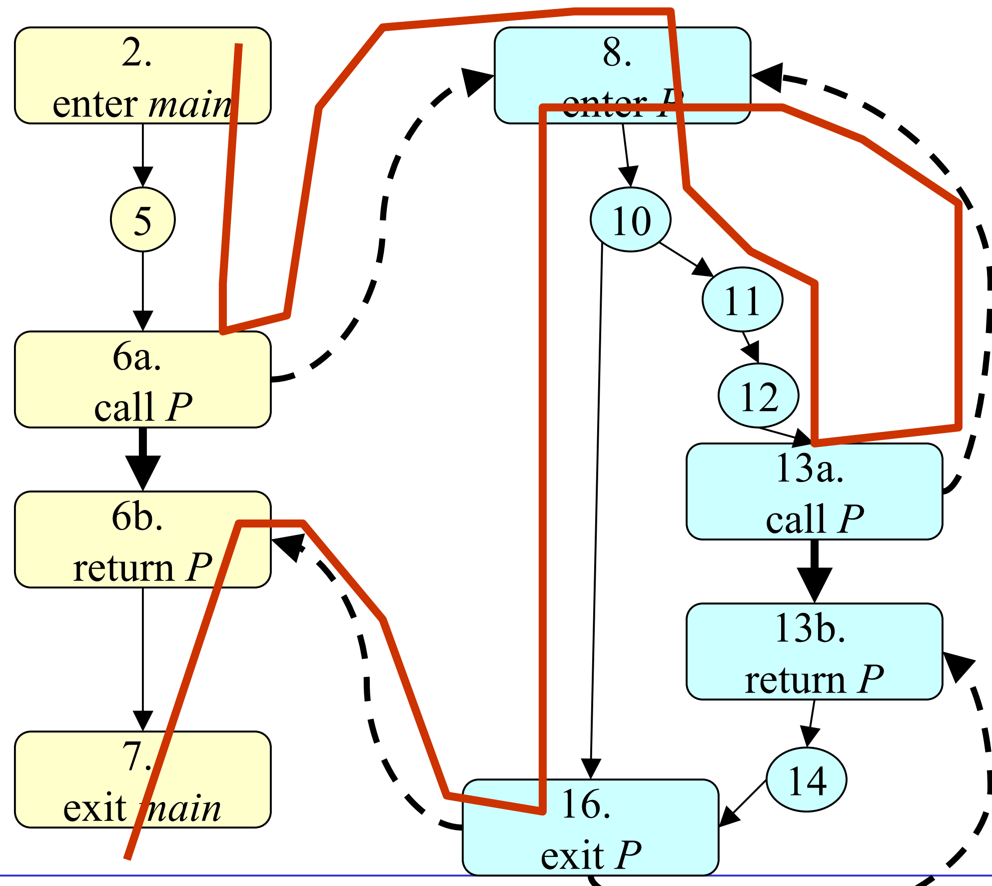
Caminhos válidos

- Caminhos válidos em um GFCI são :
  - sintaticamente possíveis,
  - semânticamente válidos



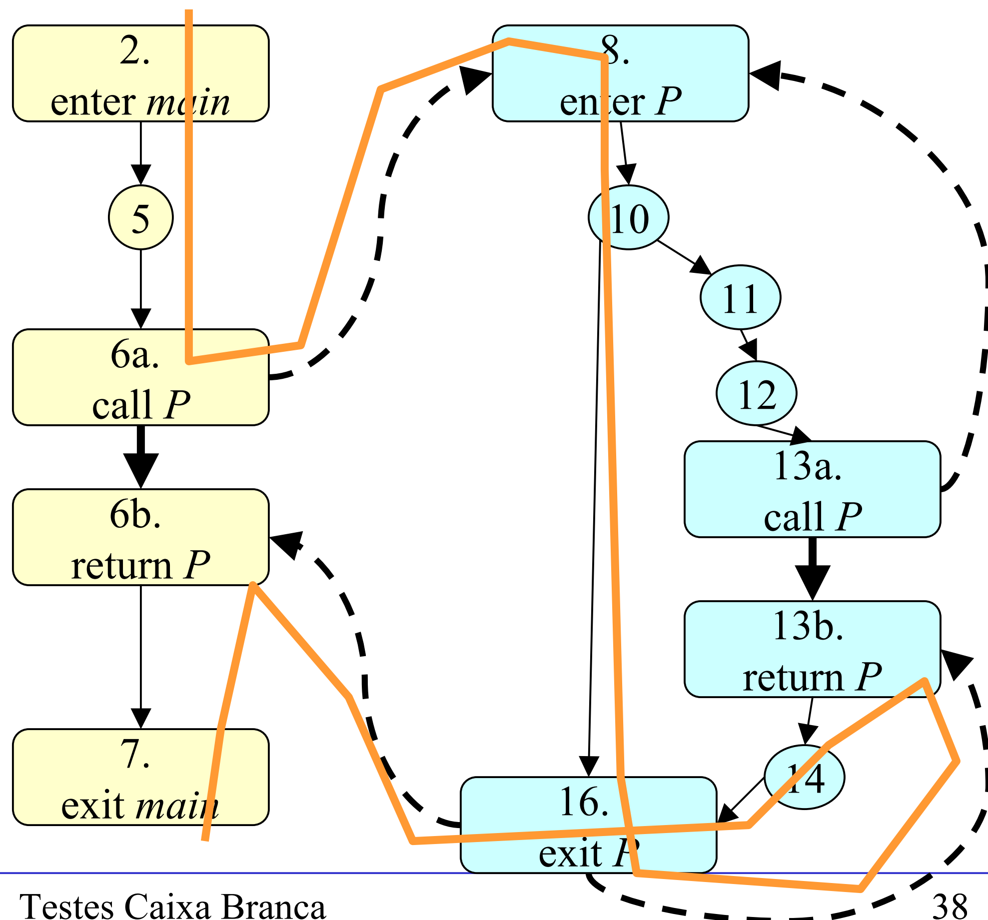
Caminhos inválidos (1)

- Caminhos inválidos em um GFCI são :
  - sintaticamente possíveis,
  - semânticamente inviáveis



Caminhos inválidos (2)

- Caminhos inválidos em um GFCI são :
  - sintaticamente possíveis,
  - semânticamente inviáveis



Considerações sobre GFCI

☺ Simples de construir

☺ Critérios de cobertura de GFCs podem ser aplicados

😐 Orientação a objetos:

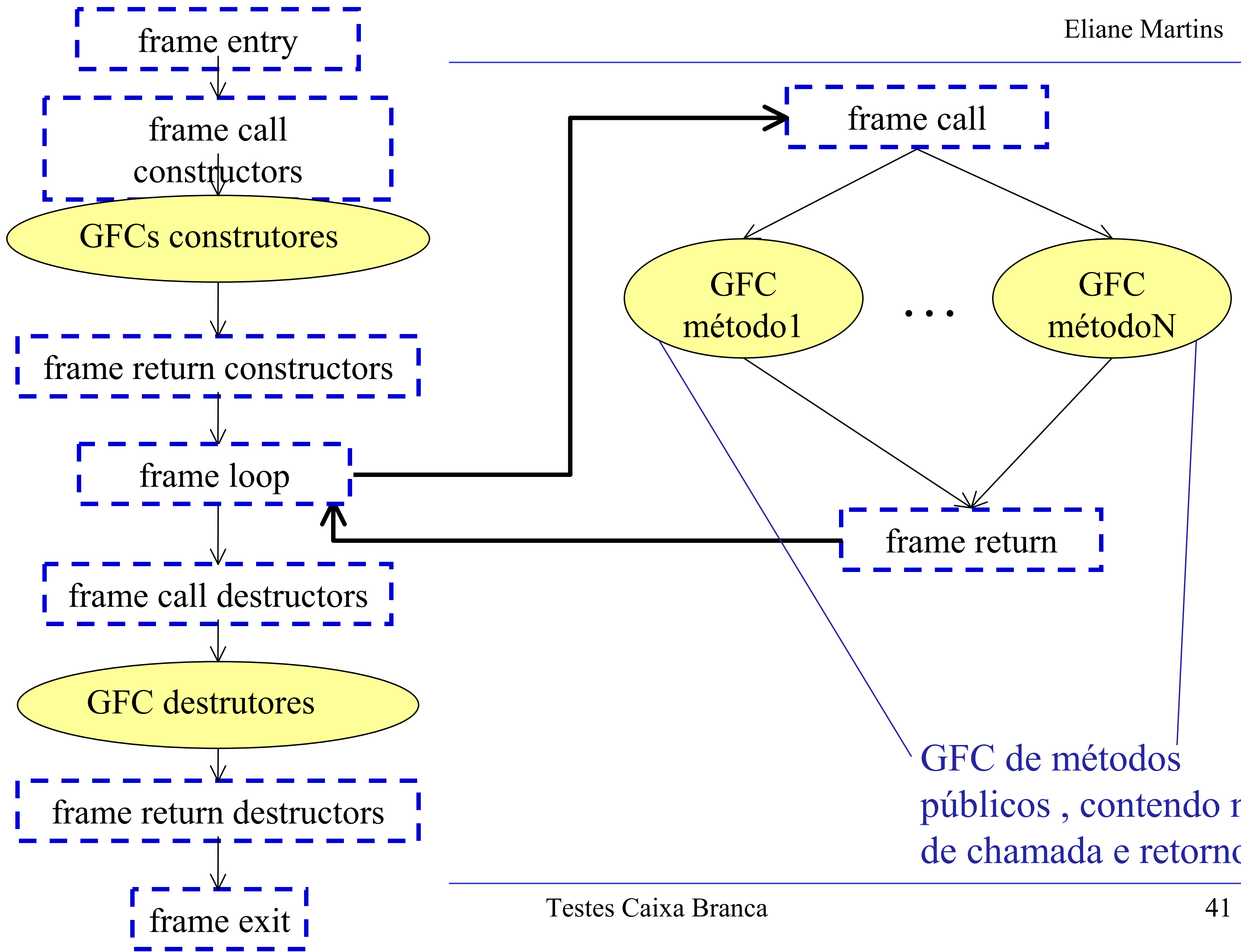
- Para algumas classes:
  - os métodos não chamam uns aos outros
  - os métodos podem ser executados em uma ordem arbitrária
- Exemplo:
- Como estabelecer um fluxo de controle entre os métodos da classe pilha?

Classe pilha
push( ) pop( ) isEmpty( )

Grafo de Fluxo de Controle para Classes

- Proposto por Rothermel e Harrold (1994) para classes em C++
- Como o GFCI, tb é formado por uma coleção de GFCs para cada método
- Como o GFCI, também interliga chamadas de métodos
- Noção de *frame*, para simular chamadas a métodos (*driver*), o que acrescenta 9 nós a mais ao GFCI:
  - *Frame entry* e *frame exit*: representam entradas e saídas de um frame
  - *Frame call constructors, frame return constructors*: representam chamadas (e retorno) aos construtores
  - *Frame call destructors, frame return destructors*: representam chamadas (e retorno) aos destrutores
  - *Frame loop*: contém as chamadas aos métodos públicos
  - *Frame call* e *frame return*: representam chamadas a métodos públicos que não são construtores e nem destrutores

[Rothermel et al 1999]


frame entry
frame call constructors
GFCs construtores
frame return constructors
frame loop
frame call destructors
GFC destrutores
frame return destructors
frame exit
frame call
GFC método1
. . .
GFC métodoN
frame return
GFC de métodos públicos , contendo nós de chamada e retorno

Considerações sobre GFCI

☺ Simples de construir

☺ Critérios de cobertura de GFCs podem ser aplicadas

😐 Orientação a objetos:

- Como fica o tratamento de classes quando os métodos podem ser usados em uma ordem arbitrária?
  - GFCC – grafo de fluxo de controle para uma classe
- Como representar a interação entre várias classes?
  - Extensão do GFCC para várias classes → escalabilidade?
- Como representar o tratamento de exceções?
  - Existem algumas propostas (e.g. [Sinha e Harrold, 1998])
- Outros aspectos de OO: polimorfismo e ligação dinâmica, objetos passados como parâmetros

Grafo de Fluxo de Dados Interprocedimental

- Fluxo de dados entre chamadas de procedimentos ou métodos é mais complicado do que fluxo de controle:
  - Relação entre parâmetro formal e parâmetro real:
    - Valores passados como parâmetros podem mudar de nome
  - Diferentes formas de compartilhamento de dados
  - Achar os usos que possam ser alcançados por um def é difícil
  - Diversas propostas:
    - Harrold e Soffa (1989)
    - Repps et al (1995)
    - Amman e Offutt (2008)
    - ...

Análise de acoplamento entre módulos de Amman e Offut

1. **declare** *g*: **integer**
2. **program** *main*
3. **begin**
4. **declare** *x*: **integer**
5. **read**(*x*) ← **última def**
6. **call** *P* (*x*) ← **(ponto de) chamada**
7. **end**

8. **procedure** *P* (**value** *a* : **integer**)
9. **begin**
10. **if** (*a* > 0) **then** ← **primeiro uso**
11. **read**(*g*)
12. *a* := *a* - *g* ← **última def**
13. **call** *P* (*a*)
14. **print**(*a*, *g*) ← **primeiro uso**
15. **fi**
16. **end**

[Reps et al. 1995]

Alguns conceitos

- Última-def: conjunto de nós que definem uma variável x e para os quais existe um caminho livre de definição entre um ponto de chamada e o uso em outro módulo
- Primeiro-uso: conjunto de nós que têm uso de y e para os quais existe um caminho livre de definição e de uso entre
  - o ponto de entrada no módulo (se o uso é no módulo chamado) e esses nós ou
  - o ponto de chamada (se o uso é no módulo chamador) e esses nós
- Acoplamento-DU: acoplamento entre última-def e primeiro-uso

Exemplo – Ammann e Offutt

```
1 // Cálculo da raiz de equação do 2º grau
2 import java.lang.Math;
3
4 class Quadratic
5 {
6  private static float Root1, Root2;
7
8  public static void main (String[] argv)
9  {
10     int X, Y, Z;
11     boolean ok;
12     int controlFlag = Integer.parseInt (argv[0]);
13     if (controlFlag == 1)
14     {
15         X = Integer.parseInt (argv[1]);
16         Y = Integer.parseInt (argv[2]);
17         Z = Integer.parseInt (argv[3]);
18     }
19     else
20     {
21         X = 10;
22         Y = 9;
23         Z = 12;
24     }
25         ok = Root (X, Y, Z);
26         if (ok)
27           System.out.println
28               ("Quadratic: " + Root1 + Root2);
29         else
30             System.out.println ("No Solution.");
31  }
32
33 // Obtém os coeficientes e calcula a raiz
34  private static boolean Root (int A, int B, int C)
35  {
36     float D;
37     boolean Result;
38     D = (float) Math.pow ((double)B,
           (double2-4.0)*A*C );
39     if (D < 0.0)
40     {
41        Result = false;
42        return (Result);
43     }
44     Root1 = (float) ((-B + Math.sqrt(D))/(2.0*A));
45     Root2 = (float) ((-B – Math.sqrt(D))/(2.0*A));
46     Result = true;
47     return (Result);
48  } / /End method Root
49
50  } // End class Quadratic
```

```
1 // Cálculo da raiz de equação do 2º grau
2 import java.lang.Math;
3
4 class Quadratic
5 {
6  private static float Root1, Root2;
7
8  public static void main (String[] argv)
9  {
10      int X, Y, Z;
11     boolean ok;
12     int controlFlag = Integer.parseInt (argv[0]);
13     if (controlFlag == 1)
14     {
15        X = Integer.parseInt (argv[1]);
16        Y = Integer.parseInt (argv[2]);
17        Z = Integer.parseInt (argv[3]);
18     }
19     else
20     {
21        X = 10;
22        Y = 9;
23        Z = 12;
24     }
```

Variáveis globais

últimas-defs

```
25          ok = Root (X, Y, Z);      <-- Ponto de chamada
26          if (ok)
27            System.out.println
28               ("Quadratic: " + Root1 + Root2);
29          else
30             System.out.println ("No Solution.");
31     }
32
33 // Obtém os coeficientes e calcula a raiz
34     private static boolean Root (int A, int B, int C)
35     {
36        float D;
37        boolean Result;
38        D = (float) Math.pow ((double)B,
              (double2-4.0)*A*C );
39        if (D < 0.0)
40        {
41           Result = false;
42           return (Result);
43        }
44        Root1 = (float) ((-B + Math.sqrt(D))/(2.0*A));
45        Root2 = (float) ((-B – Math.sqrt(D))/(2.0*A));
46        Result = true;
47        return (Result);
48     } / /End method Root
49
50     } // End class Quadratic
```

primeiro-uso no ponto de chamada

primeiro-uso no código chamado

última-def

última-def

© Ammann & Offutt

Requisitos de teste – Acoplamento DU

Descrição dos pares: (método, variável, comando)

(main (), X, 15) – (Root (), A, 38)

(main (), Y, 16) – (Root (), B, 38)

(main (), Z, 17) – (Root (), C, 38)

(main (), X, 21) – (Root (), A, 38)

(main (), Y, 22) – (Root (), B, 38)

(main (), Z, 23) – (Root (), C, 38)

(Root (), Root1, 44) – (main (), Root1, 28)

(Root (), Root2, 45) – (main (), Root2, 28)

(Root (), Result, 41) – (main (), ok, 26)

(Root (), Result, 46) – (main (), ok, 26)

Considerações sobre a técnica de Amman & Offutt

- Considera somente variáveis **usadas ou definidas** no **módulo chamado**
  - **Últimas-def** que não correspondem a nenhum **primeiro-uso** não são de interesse
- Leva em conta as **inicializações implícitas** de classes e variáveis globais
- Não leva em conta **transitividade** do acoplamento DU, pois é caro
  - Se A chama B e B chama C, não acopla últimas-defs de A com primeiros-usos em C
- **Vetores e demais variáveis estruturadas**: referência a um elemento é considerada referência a todos os elementos

Exercício 1

- Crie o grafo de chamadas para o programa dado.

```
1  procedure f()
2  begin
3      call g()
4      call g()
5      call h()
6  end
7  procedure g()
8  begin
9      call h()
10     call i()
11 end
12 procedure h()
13 begin
14 end
15 procedure i()
16     procedure j()
17     begin
18     end
19 begin
20     call g()
21     call j()
22 end
```

Exercício 2

- Dado o fragmento de programa ao lado:
  - Obtenha o grafo de chamadas para esse fragmento
  - Dado o conjunto de testes a seguir:
    - T= {t1 = f1(0,0); t2 = f1(1, 1); t3 = f1(0, 1); t4 = f1(3, 2)}
    - T cobre o critério todos os nós para o grafo de chamada?
    - T cobre o critério todas as arestas?
  - Obtenha o grafo de fluxo de controle inter-procedimental
    - Para cada caso de teste de T, indique os caminhos seguidos nesse grafo

```
public static void f1 (int x, int y)
{
    if (x<y) {f2(y);}
     else { f3(y);}
}
public static void f2 (int a)
{
    if (a%2==0) {f3(2*a);}
}
public static void f3 (int b)
{
    if (b>=0) {f4( );} else {f5( );}
}
public static void f4 ( ) { … f6( ) …}
public static void f5 ( ) { … f6( ) …}
public static void f6 ( ) { … }
```

© Ammann & Offutt

Exercício 2a

- Dado o fragmento de programa ao lado:
  - Obtenha o grafo de chamadas para esse fragmento
  - Dado o conjunto de testes a seguir:
    - T= {t1 = f1(0,0); t2 = f1(1, 1); t3 = f1(0, 1); t4 = f1(3, 2)}
    - T cobre o critério todos os nós para o grafo de chamada?
    - T cobre o critério todas as arestas?
  - Qual a ordem de testes de integração que geraria menos stubs?
  - Qual a ordem de testes de integração que exige menos drivers?

```
public static void f1 (int x, int y)
{
    if (x<y) {f2(y);}
     else { f3(y);}
}
public static void f2 (int a)
{
    if (a%2==0) {f3(2*a);}
}
public static void f3 (int b)
{
    if (b>=0) {f4( );} else {f5( );}
}
public static void f4 ( ) { … f6( ) …}
public static void f5 ( ) { … f6( ) …}
public static void f6 ( ) { … }
```

© Ammann & Offutt

Exercício 3

- Use os procedimentos dados e responda às questões abaixo:
  - Obtenha o grafo de fluxo de controle inter-procedimental
  - Identifique os pontos de chamada no grafo
  - Identifique os pares última-def e primeiro-uso
  - Crie um conjunto de testes que satisfaça ao critério *todos-acoplamentos-DU*

```
public static void trash (int x)
{
    int m, n;
    m = 0;
    if (x>0) m = 4;
    if (x>5) {n=3*m)} else { n=4*m;}
    int o = takeOut (m, n);
    System.out.print (“o is:”+ o);
}
public int takeOut (int a, int b)
{
     int d, e;
     d = 42*a;
     if (a>0) e = 2*b + d;
           else e = b + d;
     return (e);
}
```

© Ammann & Offutt

Sumário

- Principais pontos:
  - –
  - –
  - –
  - –