SSC0721 – Teste e inspeção de software

Teste de Mutação

Prof. Marcio E. Delamaro

delamaro@icmc.usp.br

Teste baseado em defeitos

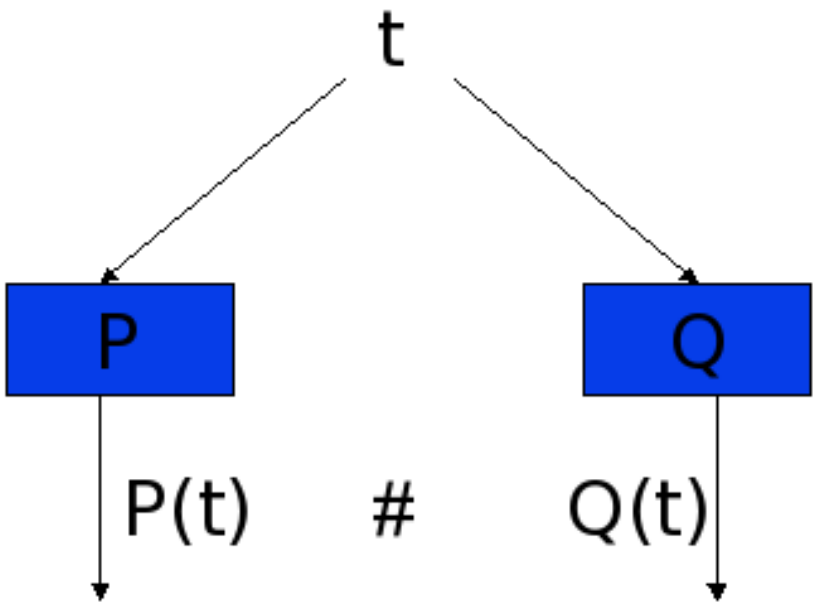
- Utiliza informação sobre defeitos comuns
- Teste de mutação é o critério mais conhecido
- Error seeding

Error seeding

- Defeitos são adicionados ao programa de maneira uniforme
- Casos de teste devem revelar tanto defeitos naturais quanto defeitos semeados
- Proporção de defeitos semeados que são revelados dá idéia da proporção de defeitos naturais revelados
- Lago dos peixes vermelhos

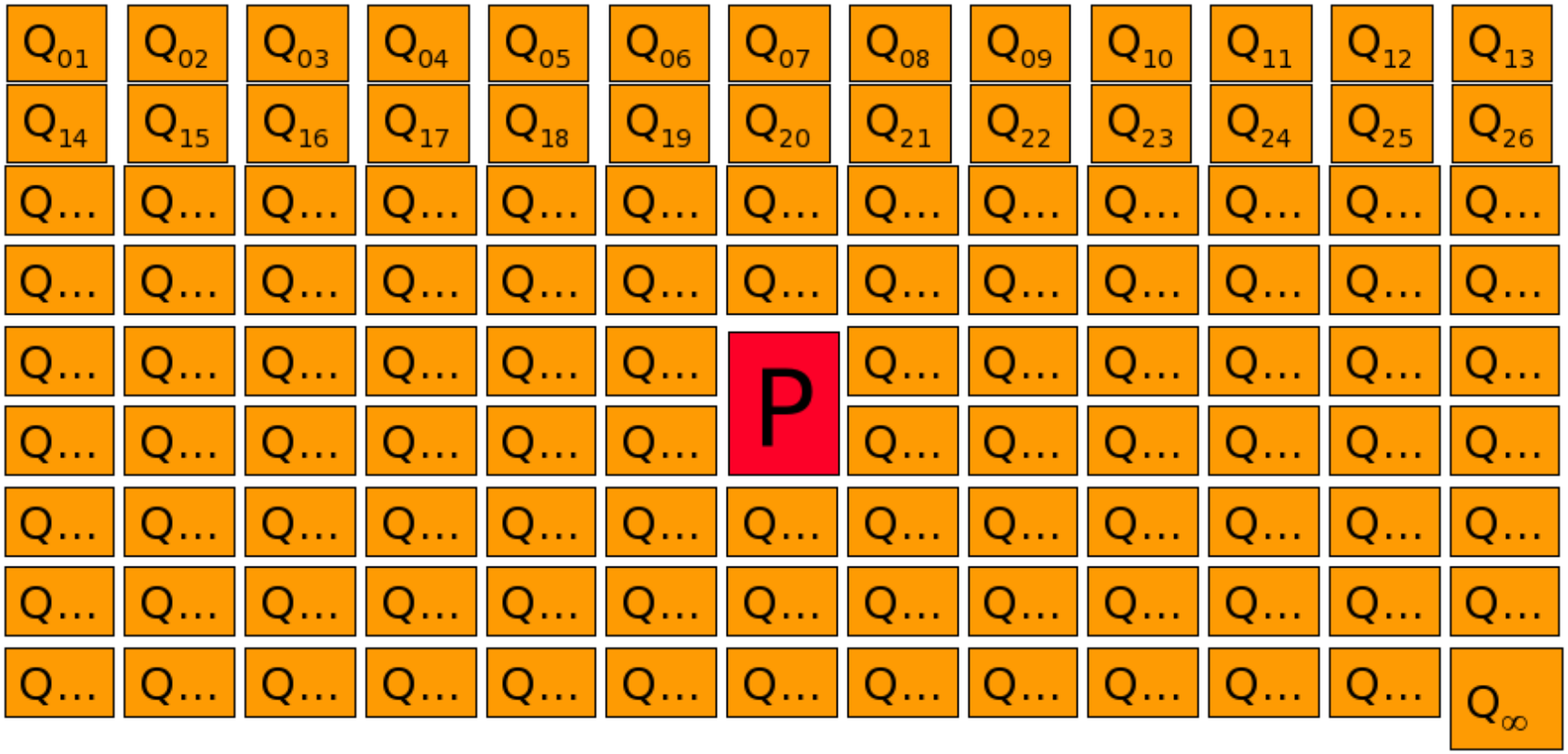
Teste de mutação

- A idéia por trás do teste de mutação é mostrar que o programa em teste não possui determinados tipos de defeitos



Correção absoluta

- Tomando todos os programas Q, é possível provar a correção de P

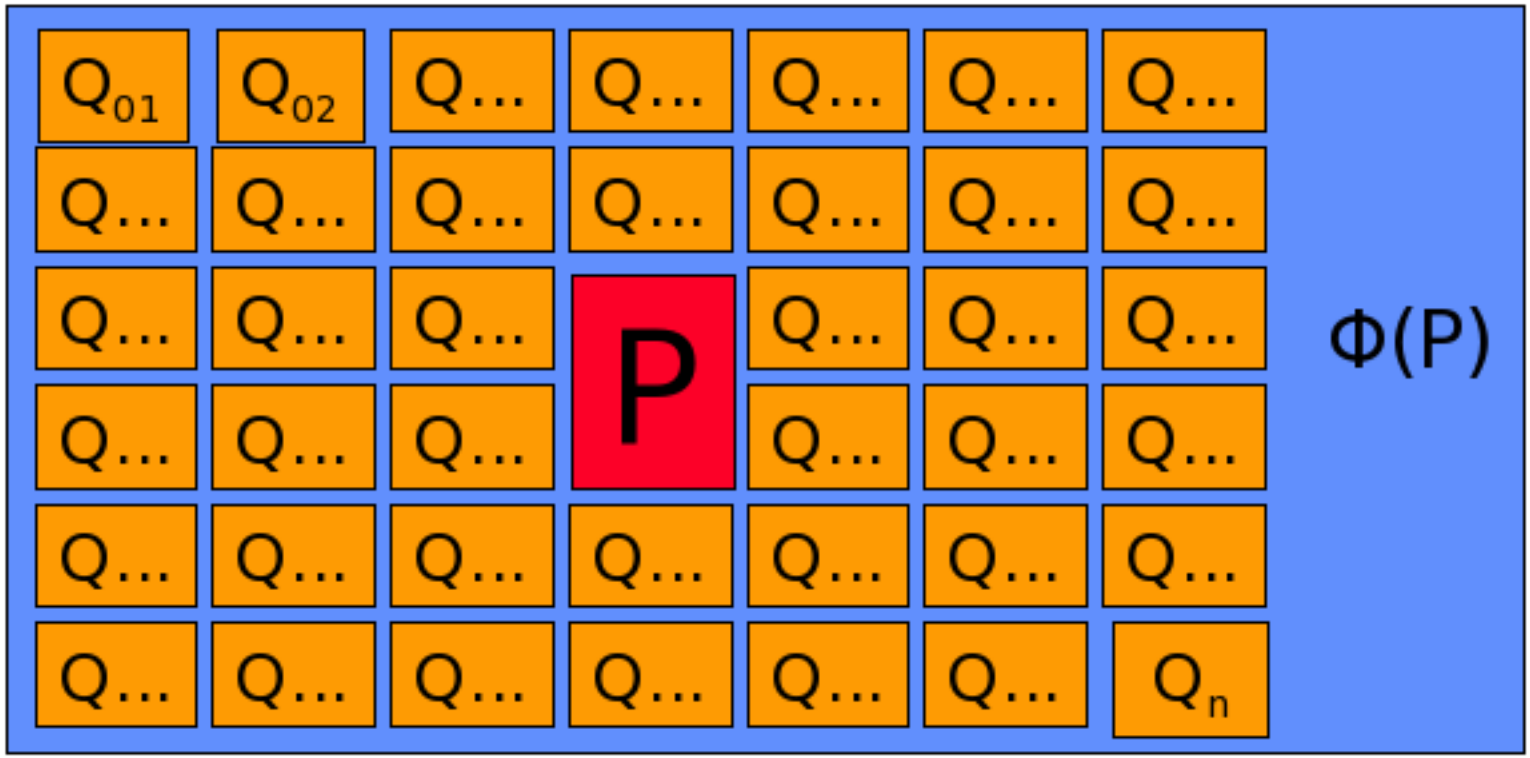


Correção relativa

- O problema com esta vizinhança é que ela é infinita
- Torna-se impossível executar e comparar cada programa Q_i
- Estabelece-se então uma vizinhança $\Phi(P)$ que contém apenas um conjunto finito de programas
- Esses programas contêm pequenos desvios sintáticos que representam defeitos simples

Correção relativa

- Casos de teste que distingam esses mutantes mostram que P está livre de determinados tipos específicos de defeitos

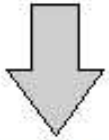


Hipóteses

- Hipótese do programador competente
  - P está correto ou próximo do correto
- Efeito de acoplamento
  - Casos de teste capazes de distinguir mutantes que possuem pequena diferença semântica em relação a P devem também revelar outros tipos de defeitos

Hipóteses

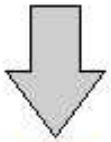
```
read x, y, z
m := x
if m < y
    m := y
if m < z
    m := z
print m
```



4

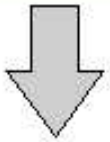
```
X = 4
Y = 2
z = 1
```

```
read x, y, z
m := x
if m ≠ y
    m := y
if m < z
    m := z
print m
```



2

```
read x, y, z
m := y
if m > z
    m := z
print m
```



1

Operadores de mutação

- Os operadores de mutação determinam o tipo de alteração sintática que deve ser feita para a criação dos mutantes
- Esses operadores buscam introduzir pequenas alterações semânticas através de pequenas alterações sintáticas que representam defeitos típicos
- Existem também os mutantes instrumentados que não modelam defeitos típicos
- Operadores dependem da linguagem alvo
  - FORTRAN 22 operadores
  - C 75 operadores

Operadores para C

- Conjunto de 75 operadores divididos em 4 classes
  - Mutação de comandos
  - Mutação de operadores
  - Mutação de variáveis
  - Mutação de constantes
- Implementados na ferramenta Proteum

Mutação de comandos

SMVB - Move Brace Up and Down

```
while (a < 10)
{
   c[a] = a+b;
   a++;
}
c[a] = 0;
```

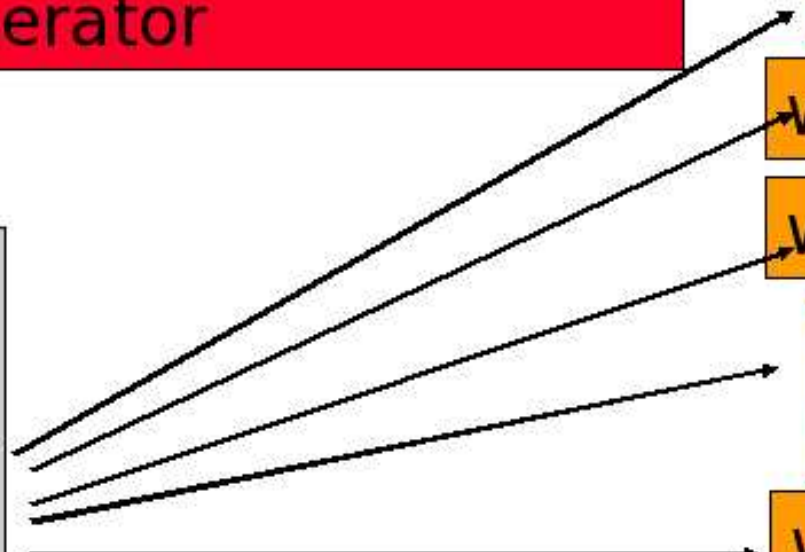
```
while (a < 10)
{
   c[a] = a+b;
}
a++;
c[a] = 0;
```

```
while (a < 10)
{
   c[a] = a+b;
   a++;
   c[a] = 0;
}
```

Mutação de operadores

ORRN - Replace relational operator by relational operator

```
while (a < 10)
{
  c[a] = a+b;
  a++;
}
c[a] = 0;
```



```
while (a > 10)
```

```
while (a <= 10)
```

```
while (a >= 10)
```

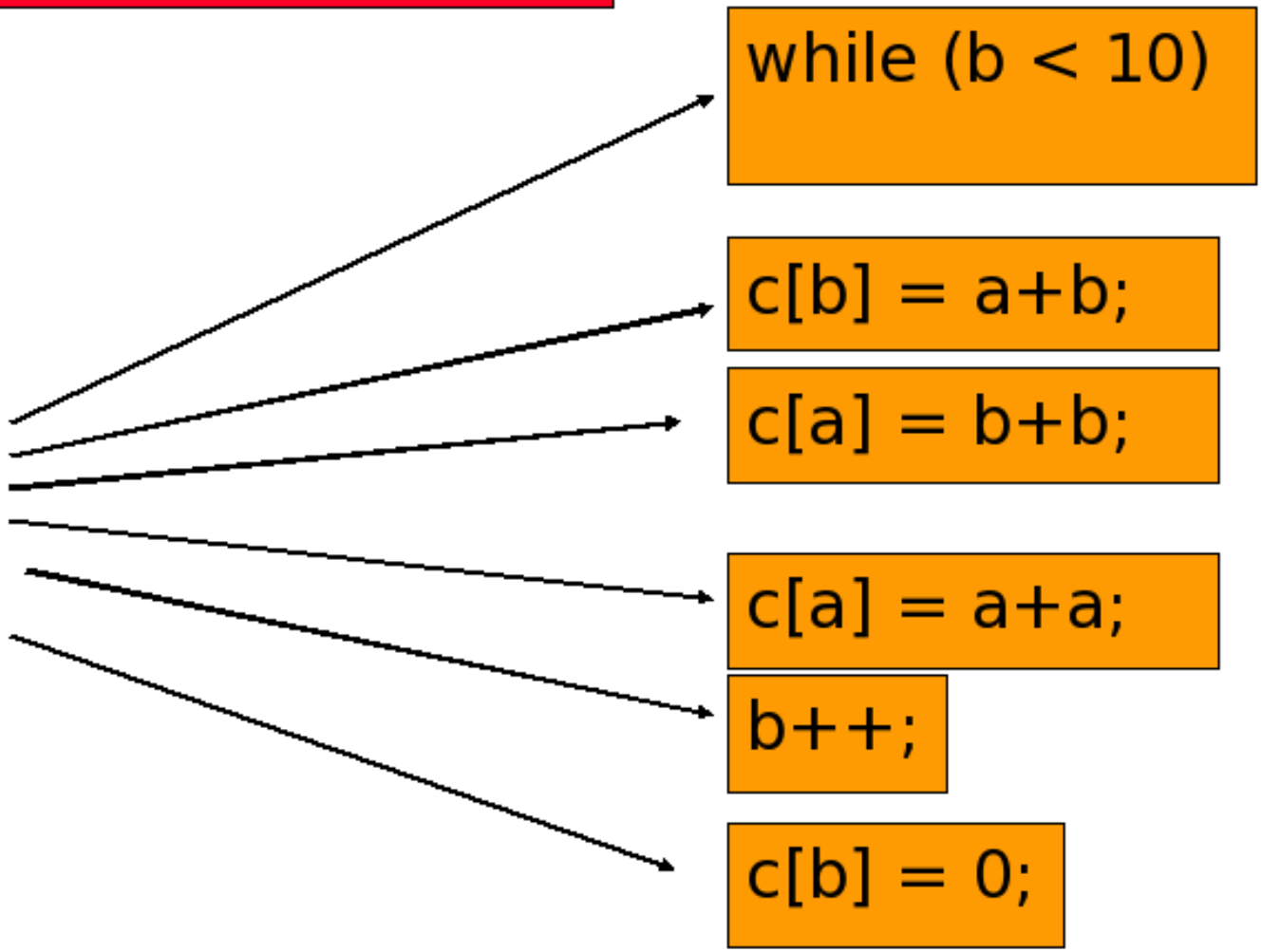
```
while (a != 10)
```

```
while (a == 10)
{
  c[a] = a+b;
  a++;
}
c[a] = 0;
```

Mutação de variáveis

Vssr - scalar variable replacement

```
while (a < 10)
{
   c[a] = a+b;
   a++;
}
c[a] = 0;
```



Mutação de constantes

Cccr - Constant for constant replacement

```
while (a < 10)
{
   c[a] = a+b;
   a++;
}
c[a] = 0;
```

```
while (a < 10)
{
   c[a] = a+b;
   a++;
}
c[a] = 10;
```

```
while (a < 0)
{
   c[a] = a+b;
   a++;
}
c[a] = 0;
```

Mutantes instrumentados

STRP - trap on statement execution

```
while (a < 10)
{
   c[a] = a+b;
   a++;
}
c[a] = 0;
```

```
while (a < 10)
{
   trap();
   a++;
}
c[a] = 0;
```

```
while (a < 10)
{
   c[a] = a+b;
   trap();
}
c[a] = 0;
```

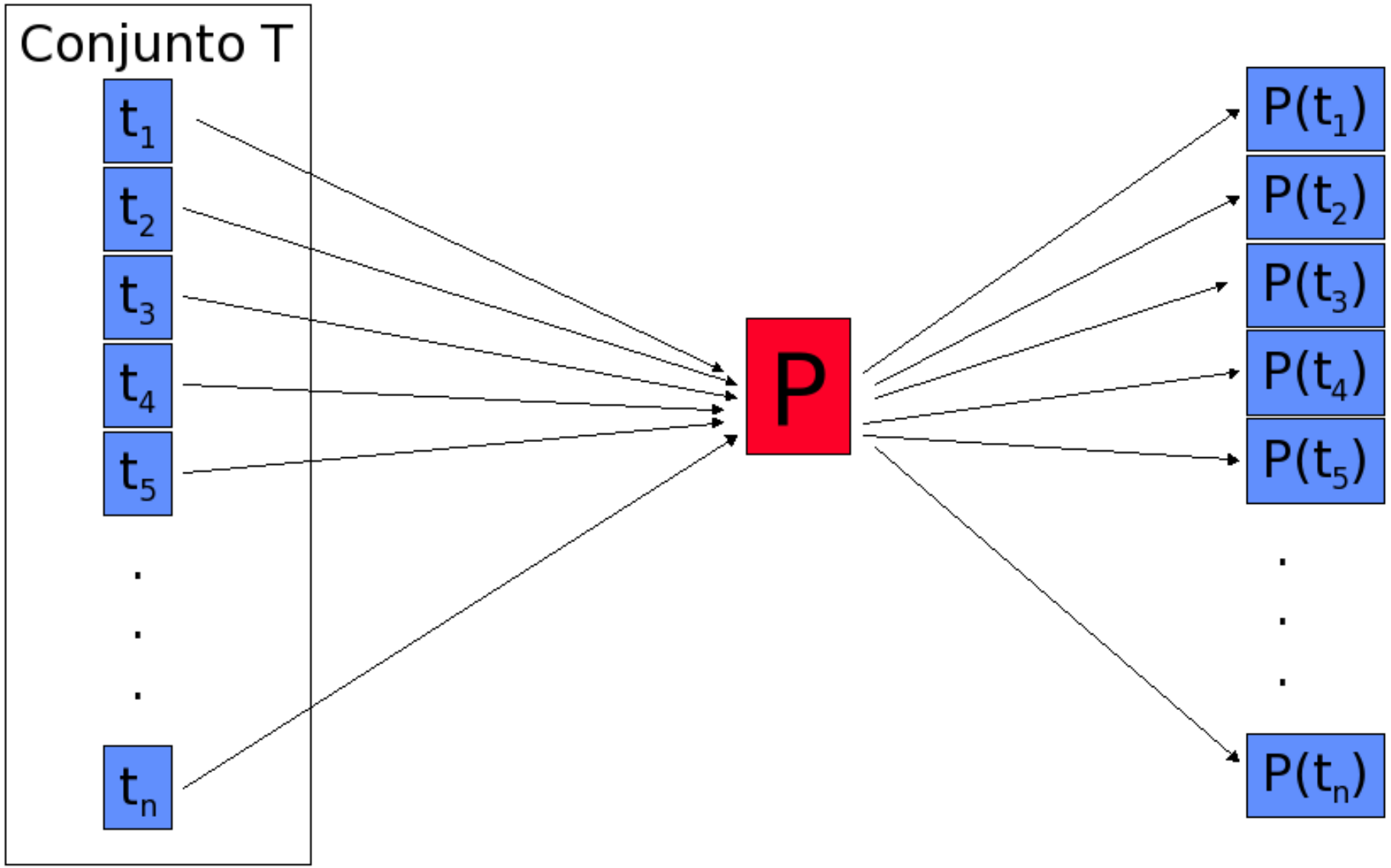
```
trap();
c[a] = 0;
```

```
while (a < 10)
{
   c[a] = a+b;
   a++;
}
trap();
```

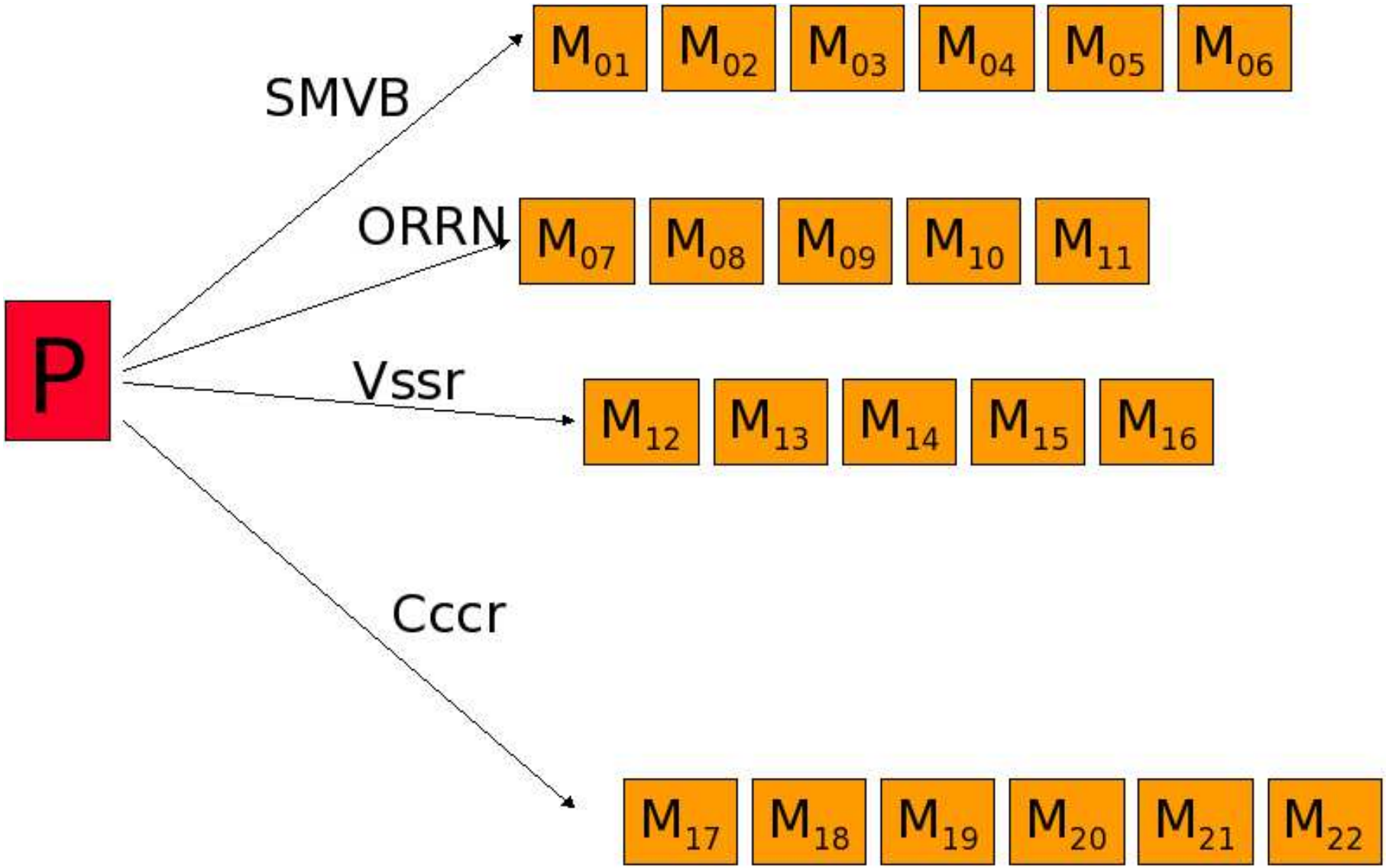
Aplicação do critério

- P é executado com os casos de teste de T
- Mutantes são gerados
- Mutantes são executados com os casos de teste de T
- Mutantes são analisados

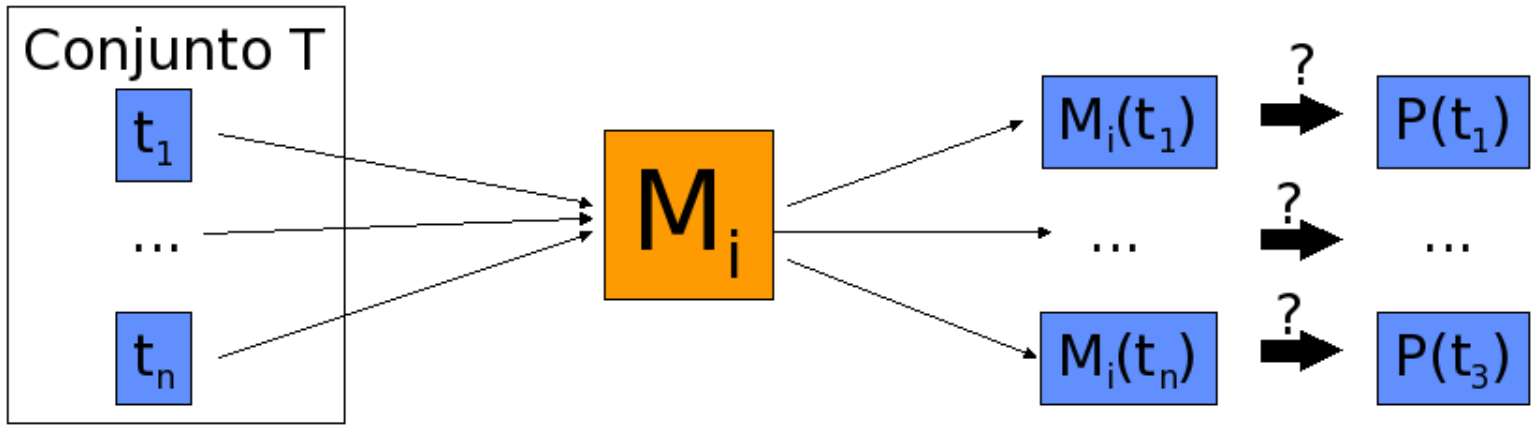
Execução de P



Geração dos mutantes

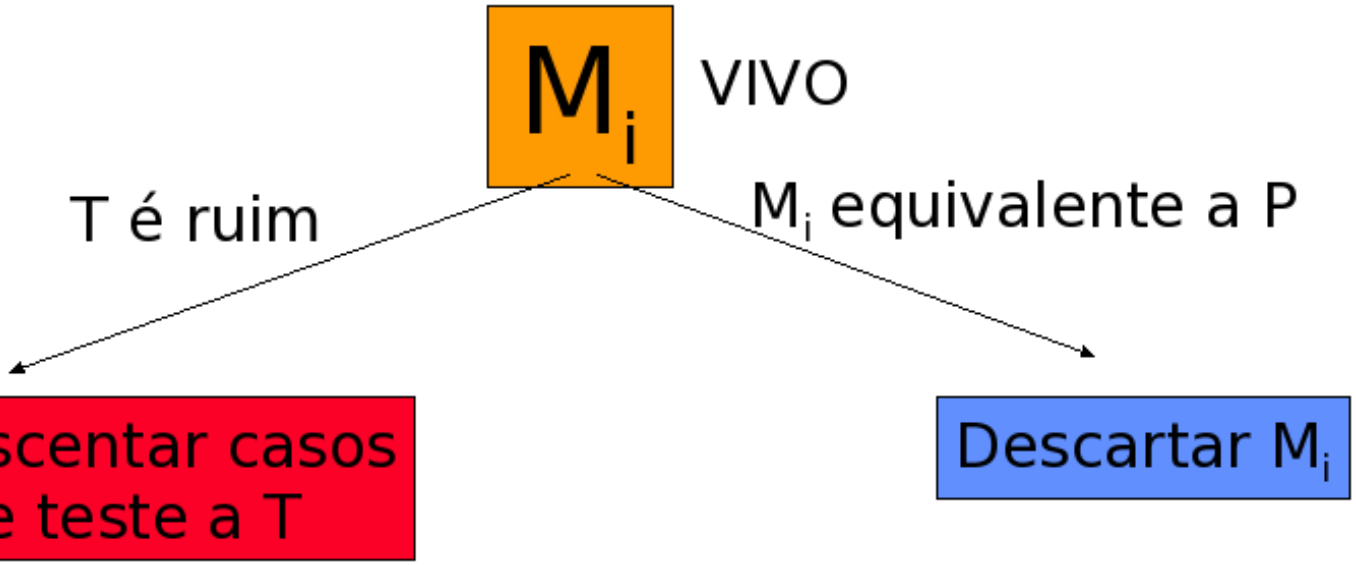


Execução dos mutantes



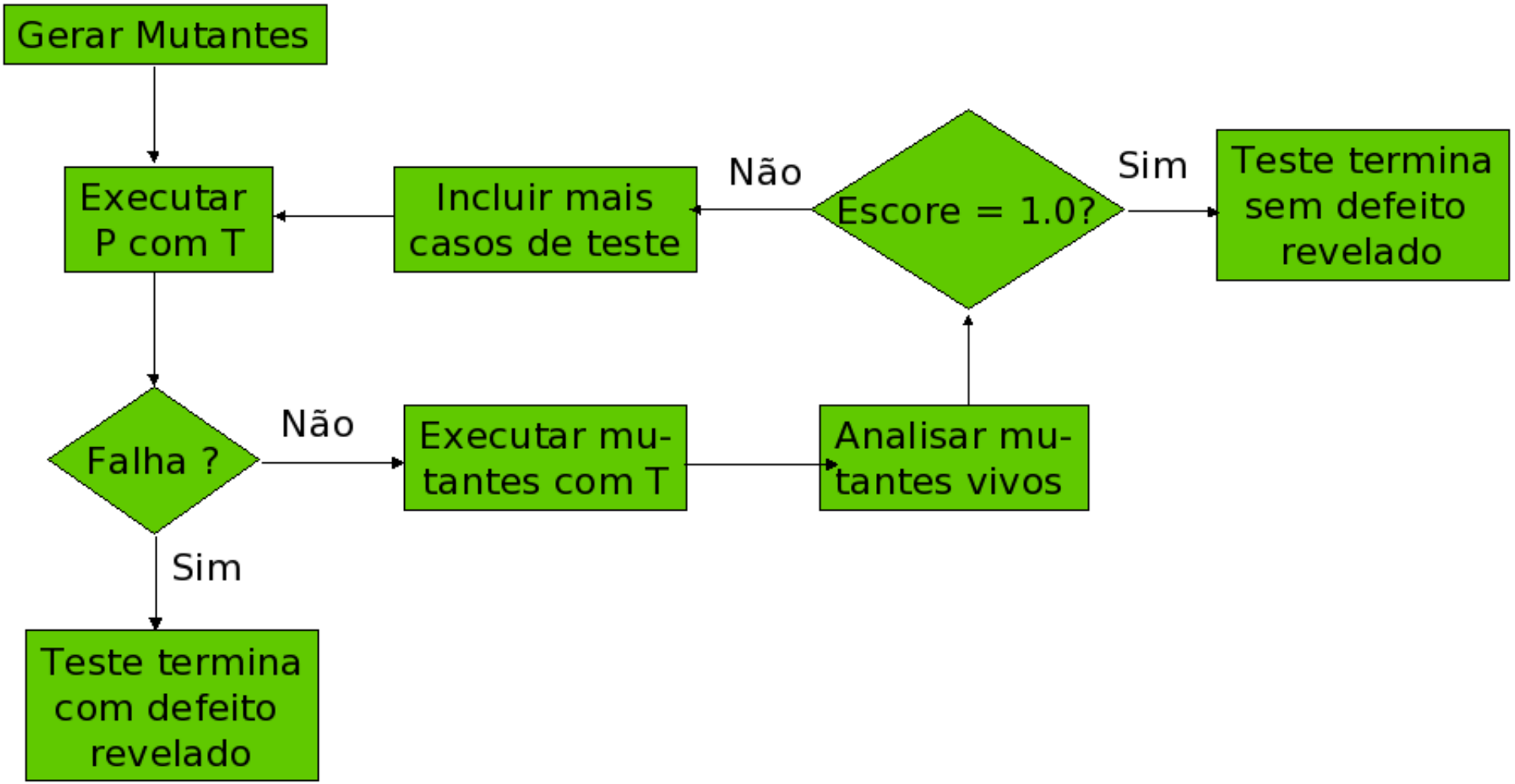
- $M_i(T) = P(T)$ então M_i está vivo
- senão M_i está morto
- escore de mutação $= \frac{\text{\# mutantes mortos}}{\text{\# total de mutantes}}$

Análise dos mutantes



- escore de mutação $= \frac{\text{\# mutantes mortos}}{\text{\# mutantes não equivalentes}}$

Aplicação do critério



Considerações

- Mutantes equivalentes
  - Assim como associações de fluxo de dados não executáveis, a equivalência é indecidível
- Custo
  - O número de mutantes a serem executados pode ser muito alto
  - Exemplo: Cal.c gera 4257 mutantes para 106 LOC

Equivalência

- Problema indecidível

```
read x, y, z
m := x
if m < y
    m := y
if m < z
    m := z
print m
```

```
read x, y, z
m := x
if m <= y
    m := y
if m < z
    m := z
print m
```

Custo

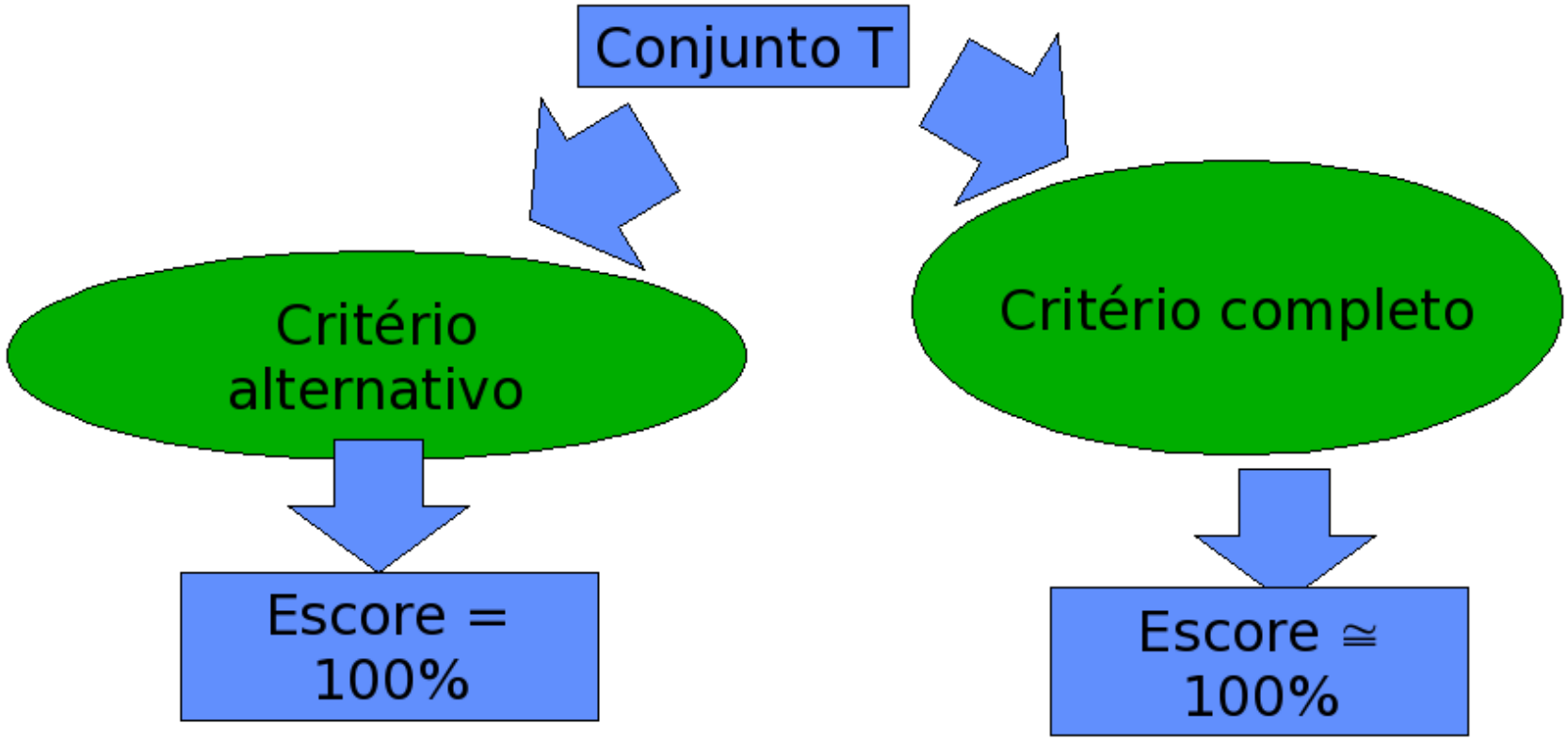
- Uma das soluções para o problema de custo envolve a utilização de hardware de alta performance ou paralelo.
- Nesse caso, o custo em termos de tempo diminui mais o custo total do teste é alto
- Soluções são mais teóricas do que práticas

Redução dos mutantes

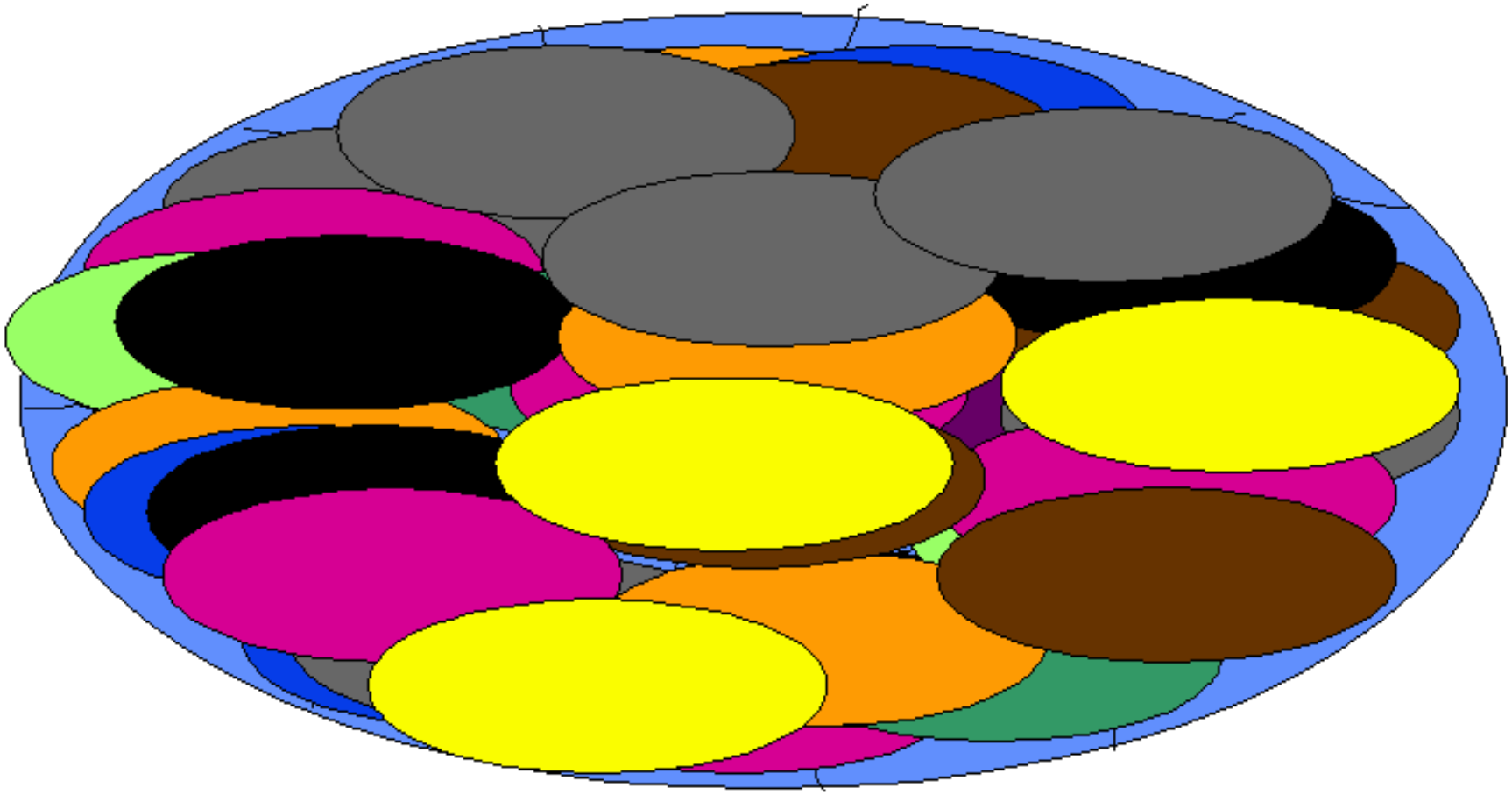
- Outro tipo de solução tenta reduzir o número de mutantes através de algumas abordagens
- Critérios de mutação alternativos
  - Mutação restrita
  - Mutação randômica
- Estratégias podem ser combinadas

Mutação alternativa

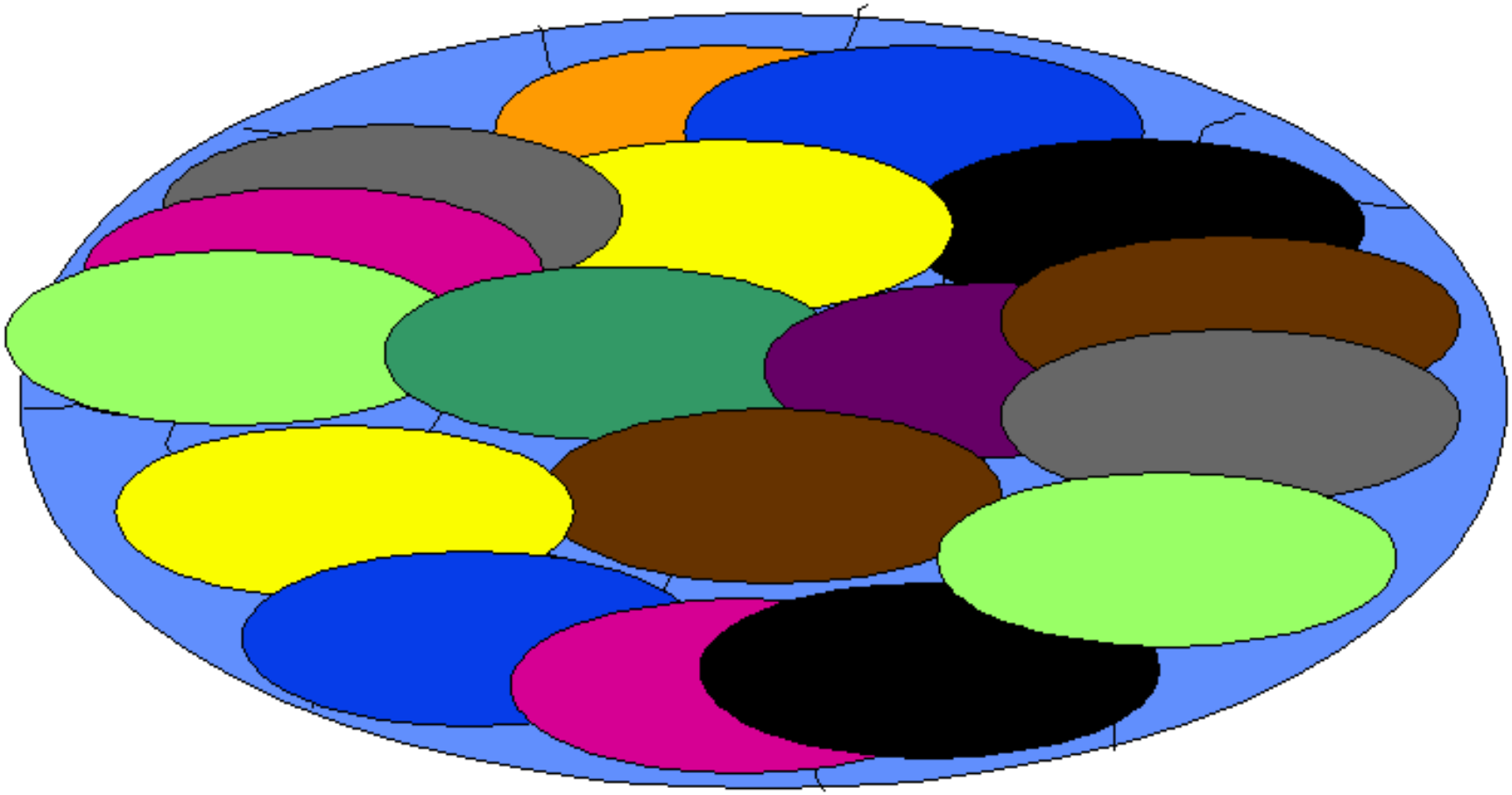
- Utilizando critérios alternativos obtém-se conjuntos quase tão adequados em relação ao critério completo



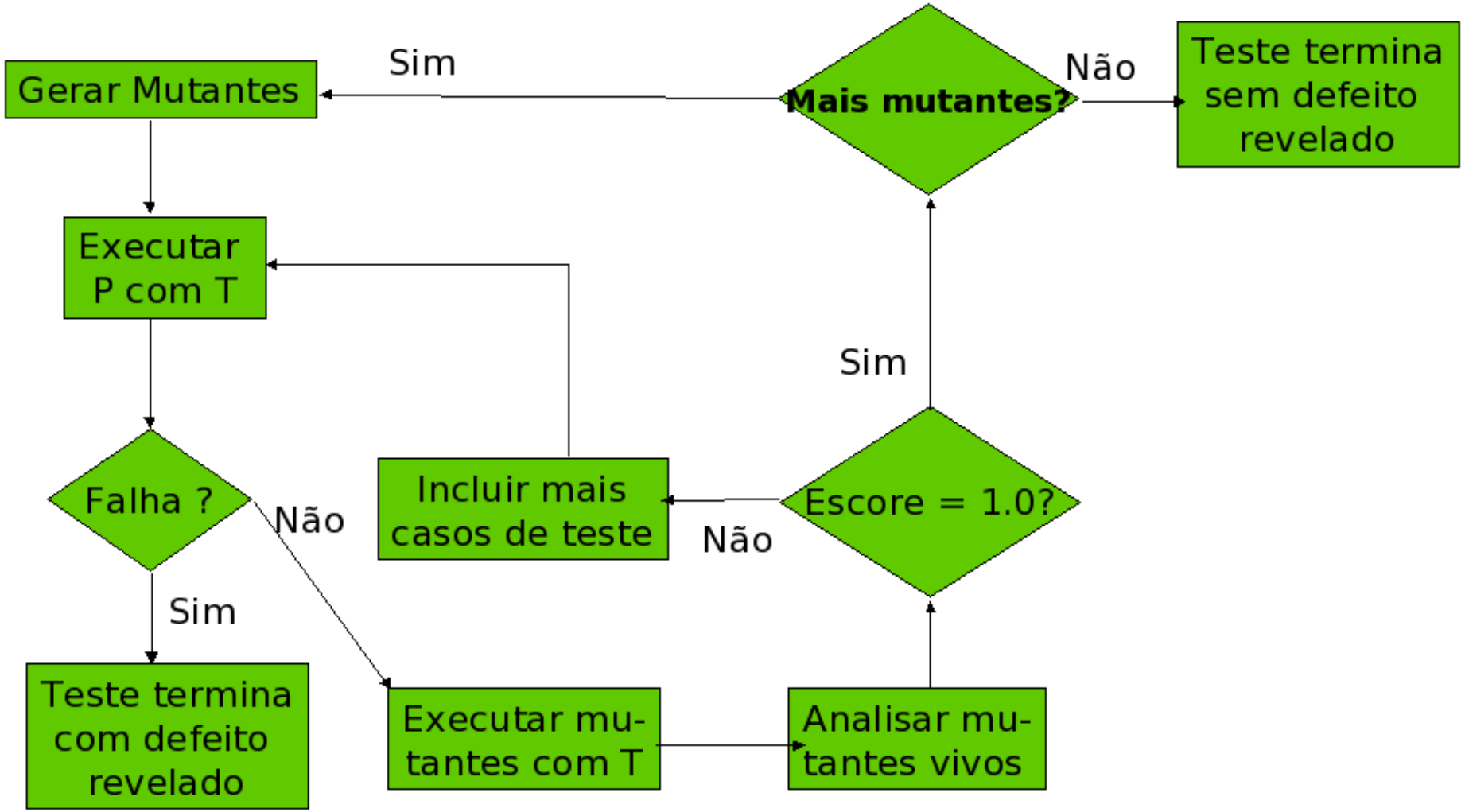
Mutação alternativa



Mutação alternativa



Estratégia incremental



Teste de modelos

- Uma das vantagens do teste de mutação é que ele não requer a identificação de determinadas estruturas como as de fluxo de controle ou de fluxo de dados
- Por isso ele pode ser aplicado em outros tipos de “entidades”
  - Máquinas de estado finito
  - Redes de Petri
  - Statecharts

Máquina de estados finitos

Comparação com fluxo de dados

	Mutação	Fluxo de dados
Requisitos	Mutantes	Associações def-uso
Indecidibilidade	Equivalentes	Não executáveis
Custo	Alto	Menor
Inclusõa	Incomparáveis	
Aplicado a	Programas e outros	Programas

Mutação de interface

- Teste de mutação tem sido utilizado no nível de unidade
  - Operadores exercitam aspectos algoritmicos da unidade
- Mutação de interface visa as interações entre unidades
  - chamadas de funções
  - passagem de parâmetros
  - valores de retorno

Outros trabalhos

- Operadores essenciais
- Mutantes equivalentes
- Geração de dados de teste
- Modelos
- Domínios específicos (programas concorrentes)
- Oráculos