DIM0436

Testes de caixa branca
PathCrawler

20141113

1 Introdução

Essa aula de laboratório usa a ferramenta de geração de testes estruturais `PathCrawler`. Nos usaremos a versão online acessível no

`http://pathcrawler-online.com`

Por que automatizar testes estruturais ?

1. Atingir a cobertura de teste desejada é custoso
2. Deve ser verificado de novo após cada modificação do código
3. É difícil mostrar a infazibilidade de um objetivo de teste manualmente.

Assim, testes estruturais **automáticos** podem ser usados para:

- atingir elementos ainda não cobertos
- determinar a não-alcançabilidade de alguns deles
- com o custo adicional baixo

2 Questões

Entre cada questão abaixo, volte para a página inicial clicando no PacMan acima à esquerda.

2.1 Primeiros passos

1. Selecione o `Example1`. Clique on "Run Tests". Observe os resultados.
   - Quais são os valores de entradas ?
   - Quais são os valores de saída ? Todas as saídas possíveis são geradas ?
   - Explore a interface.
2. Selecione o `Example2`.
   - Observe o número de testes gerados, os dados de entrada.
   - O que parece ser o problema ?
   - Queremos gerar esses dados de teste ?
3. Selecione o `Example3` e clique no "Customize test parameters".
   (a) Restrinja as entradas `i, j, l` a valores positivos.
   (b) Adiciona 3 precondições não quantificadas (**não pressa Enter**)
      i. $i + j > k$
      ii. $j + k > i$
      iii. $i + k > j$
   (c) Execute o teste e verifica os resultados.
   (d) O que mudou ?
4. Selecione o `Example4` e clique no "Customize test parameters".
   (a) Verifique que `PathCrawler` ativou a pré-condição.
   (b) Execute os testes e observa o número de teste gerado.
   (c) São os mesmos tipos de dados de teste que no passo 3 ?

2.2 Especificação avançada das entradas

1. Selecione o `Example5`. Execute os testes com os parâmetros predefinidos. O que está errado com as saídas concretas ? Porque ?
2. Selecione o `Example6` e clique no "Customize test parameters".
   (a) Verifique que temos duas precondições quantificadas:
      i. $\forall i, i < 2, t1[i] \leq t1[i + 1]$

ii. $\forall i, i < 2, t2[i] \leq t2[i+1]$

(b) Execute o teste e observe os resultados.

(c) Os vetores `t1` e `t2` são ordenados ? `t3` é ordenado na saída ?

3. Selecione o `Example7` e clique no "Customize test parameters".

   (a) Verifique que as precondições implicam que os vetores `t1` e `t2` são ordenados.

      i. `l1` é o tamanho de `t1`, `l2` o tamanho de `t2`.

      ii. O domínio dos elementos é restrito, definido a $[-100, 100]$

   (b) Execute os testes.

   (c) Porque tem erros ? O que aconteceu ?

4. Selecione o `Example8` e clique no "Customize test parameters".

   (a) As dimensões de `t1, t2, t3` são especificadas

   (b) Verifique a adição das restrições

      i. $dim(t1) = l1$

      ii. $dim(t2) = l2$

      iii. $dim(t3) = l1 + l2$

   (c) Execute os testes. Quantos testes foram gerados ?

2.3 Cobertura parcial: critério *k-path*

Na presença de laços, o critério *all-path* pode gerar um número de testes infinitos. O critério *k-path* limite o critério *all-path* a ao máximo k iteração consecutivas de cada laço.

1. Selecione o `Example9` e clique no "Customize test parameters".

   (a) Verifique que a estrategia de teste seja *2-path*

   (b) Quantos casos de testes forma gerados ?

   (c) Os testes parecem corretos ?

2.4 Oráculos e depuração de programas

O papel de um oráculo é:

- examinar as entradas e saídas de cada teste

- determinar se a implementação respeita os resultados previstos (pelo oráculo)
- oferecer um veredito (sucesso/falha)

O oráculo pode vir:

- de uma outra implementação
- de uma verificação dos resultados (sem implementar o algoritmo)

1. Selecione o `Example10a` e clique no "Customize test parameters".
   (a) O oráculo parece completo ?
   (b) Execute os testes para verificar sua resposta.
2. Selecione o `Example10b` e clique no "Customize test parameters".
   (a) O oráculo parece completo ?
   (b) Execute os testes para verificar sua resposta.
3. Selecione o `Example10c` e clique no "Customize test parameters".
   (a) O oráculo parece completo ?
   (b) Execute os testes para verificar sua resposta.
   (c) O que é o erro na implementação ?

2.5 Usos avançados do teste estrutural

O gerador de teste `PathCrawler` explora a implementação e pode ser usado para:

- gerar *runtime errors* na execução do programa
- detectar anomalias na análises dos caminhos cobertos
  - variável não inicializada
  - *buffer overflow*
  - *integer overflow*
- cálculos desnecessários na implementação
- o tempo de execução de cada caminho

- detectar código morto (caminhos parciais infazíveis)

1. Selecione o `ExampleUninit` e clique no "Run Tests".
   (a) Tem erros o avisos ? Porque ?
   (b) Todos os caminhos fazíveis são cobertos ?
2. Selecione o `ExampleUC` e clique no "Customize Parameters".
   (a) Olhe o oráculo predefinido e os parâmetros.
   (b) Execute os testes
   (c) Essa implementação está correta ?
   (d) Examine os predicados dos casos onde x é diferente de 0.
   (e) Essa implementação está eficiente ?

2.6 Fim

Se você chegou aqui, pode continuar a partir do slide 43 do arquivo

`http://pathcrawler-online.com/tutorial/tutorial_2012.pdf`

Boa sorte!