Testando um problema da maratona

Aula para disciplina de Métodos Formais

Gabriela Moreira

Departamento de Ciência da Computação - DCC
Universidade do Estado de Santa Catarina - UDESC

20 de maio de 2024

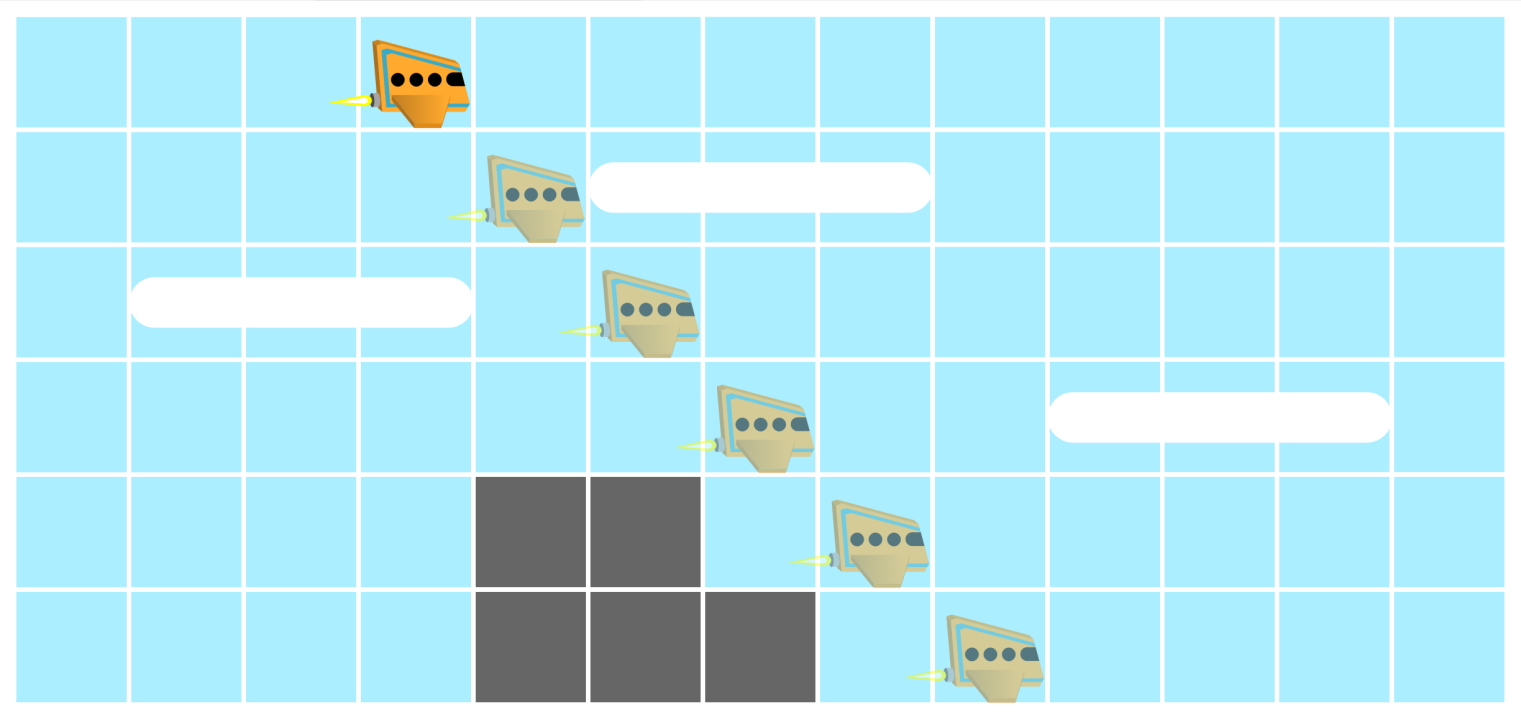
Conteúdo

Testes

Outline

Testes

Problema D: Desvio de rota



Testes comuns

- Vão testar um caso específico
- Podem ser unitários, de integração, etc
  - Existem varias classificações e debates sobre as definições precisas.

Testes comuns

- Vão testar um caso específico
- Podem ser unitários, de integração, etc
  - Existem varias classificações e debates sobre as definições precisas.
- Exemplos de entrada e saída fornecidos na prova
- Exemplos específicos que os times bolam durante a prova

Testes baseados em propriedades

Escrever propriedades é difícil, mas pode ser muito útil para testes

Testes baseados em propriedades

Escrever propriedades é difícil, mas pode ser muito útil para testes

- Dado um trajeto T de pouso da nave simulado a partir da altura h (saída do programa), e a lista H de alturas das montanhas, T[i] $>$ H[i] para todo i $<$ N
- Dado uma altura mínima h (saída do programa), não deve existir altura menor que h tal que a propriedade acima seja verdadeira.

Testes baseados em propriedades

Escrever propriedades é difícil, mas pode ser muito útil para testes

- Dado um trajeto T de pouso da nave simulado a partir da altura h (saída do programa), e a lista H de alturas das montanhas, T[i] $>$ H[i] para todo i $<$ N
- Dado uma altura mínima h (saída do programa), não deve existir altura menor que h tal que a propriedade acima seja verdadeira.

Frameworks de property-based testing (PBT) vão gerar inputs aleatórios* para o programa e, para cada um deles, ver se a propriedade é satisfeita.
*não necessariamente aleatórios. Muitos frameworks vão tentar valores extremos (zero e `MAX_INT` para inteiros, etc).

Testes baseados em propriedades

Escrever propriedades é difícil, mas pode ser muito útil para testes

- Dado um trajeto T de pouso da nave simulado a partir da altura h (saída do programa), e a lista H de alturas das montanhas, T[i] $>$ H[i] para todo i $<$ N
- Dado uma altura mínima h (saída do programa), não deve existir altura menor que h tal que a propriedade acima seja verdadeira.

Frameworks de property-based testing (PBT) vão gerar inputs aleatórios* para o programa e, para cada um deles, ver se a propriedade é satisfeita.
*não necessariamente aleatórios. Muitos frameworks vão tentar valores extremos (zero e `MAX_INT` para inteiros, etc).

Pra quem já fez TEC:

- Semelhante a decisão vs verificação

Testes baseados em propriedades II

E sobre performance?

- No cenário da maratona, os times estariam rodando esses testes na própria máquina.
- O programa em si que é submetido não inclui esses testes
- Dependendo da confiança nos testes, permite otimizações mais agressivas no código base

Testes baseados em propriedades II

E sobre performance?

- No cenário da maratona, os times estariam rodando esses testes na própria máquina.
- O programa em si que é submetido não inclui esses testes
- Dependendo da confiança nos testes, permite otimizações mais agressivas no código base

Para esse cenário da maratona, testes baseados em propriedades seriam a melhor abordagem na minha avaliação.

Testes baseados em propriedades II

E sobre performance?

- No cenário da maratona, os times estariam rodando esses testes na própria máquina.
- O programa em si que é submetido não inclui esses testes
- Dependendo da confiança nos testes, permite otimizações mais agressivas no código base

Para esse cenário da maratona, testes baseados em propriedades seriam a melhor abordagem na minha avaliação.

> *Program testing can be used to show the presence of bugs, but never to show their absence!*
> *– Edsger Dijkstra*

Testes baseados em modelos

Verificação garante que não existem bugs!

Testes baseados em modelos

Verificação garante que não existem bugs!

Mas e se, na hora de implementar o sistema especificado, eu cometer um erro?

Testes baseados em modelos

Verificação garante que não existem bugs!

Mas e se, na hora de implementar o sistema especificado, eu cometer um erro?

Model-Based Testing (MBT) é uma forma de testar se a implementação está de acordo com a especificação.

Testes baseados em modelos

Verificação garante que não existem bugs!

Mas e se, na hora de implementar o sistema especificado, eu cometer um erro?

Model-Based Testing (MBT) é uma forma de testar se a implementação está de acordo com a especificação.

No contexto do problema da maratona, um exemplo:

1. Especificamos a solução em Quint/TLA+
2. Verificamos as duas propriedades listadas - Atenção: aqui temos **verificação**, então não pode existir nenhum caso de teste danado da equipe de prova que vai nos pegar desprevindos!
   - Aqui, sabemos que nossa solução está correta. A menos, está de acordo com as propriedades.
3. Escrevemos o código C++ equivalente

Testes baseados em modelos II

Testes baseados em modelos II

Até aqui, já estamos bem fundamentados. A chance de cometermos um erro ao escrever 100 linhas de C++ é baixa. Em sistemas mais realistas, a chance de cometer um erro na implementação é bem maior.

Testes baseados em modelos II

Até aqui, já estamos bem fundamentados. A chance de cometermos um erro ao escrever 100 linhas de C++ é baixa. Em sistemas mais realistas, a chance de cometer um erro na implementação é bem maior.

Se quisermos reduzir a chance de erro de implementação, podemos usar MBT:

1. Escrevemos um adaptador que recebe uma execução (trace) e executa ela no código
   - Pra esse exemplo, temos um único input, então a execução seria somente entrada -> saída. Em outros problemas como no E: El Café, temos múltiplas "consultas" em diferentes estados do sistema (com mais ou menos quantidades de ingredientes).
2. Usamos o simulador ou model checker para produzir várias execuções a serem dadas para esse adaptador.
   - Aqui entra a parte mais legal: podemos escolher execuções interessantes usando invariantes.
     - Exemplo: Simulador, me dá uma execução aí onde a última montanha é a mais alta de todas
     - Quantas vezes na maratona vocês já construíram inputs na mão a fim de provocar uma certa situação?

Extração de código

Alguns métodos formais, como Coq, permitem extração automática da implementação a partir da especificação.

- No meu TCC, eu implementei um gerador de código Elixir a partir de TLA+.
  - Mas ele tá longe de ter a mesma confiança que o do Coq.

Trabalho 2

Nosso trabalho 2 será sobre testes baseados em modelos!

- É o maior ponto de contato entre métodos formais e indústria.
- Eu vou fazer boa parte do trabalho, vocês só vão precisar encaixar as pecinhas e ver funcionando.