Exercícios de Alloy

Alcino Cunha

22 de Outubro de 2010

1. Considere o seguinte modelo para representar os leilões em curso numa conhecida leiloeira online:

```
open util/ordering[Bid]

sig Client, Product, Bid {}
sig Auction {
  product : Product
}
sig Ebay {
  clients : set Client,
  auctions : Client -> Auction,
  bids : Client -> Auction -> Bid
}

pred A [e : Ebay] {
  e.auctions in e.clients -> Auction
  e.bids in e.clients -> Client.(e.auctions) -> Bid
}
pred B [e : Ebay] {
  all disj a,b : e.clients | no e.auctions[a] & e.auctions[b]
}
pred C [e : Ebay] {
  no e.bids.Bid.product & e.auctions.product
}
pred NoEqualBids[e : Ebay] {...}

pred Inv [e : Ebay] {
  A[e] && B[e] && C[e] && NoEqualBids[e]
}

pred IsCurrentWinner [e : Ebay, c : Client, p : Product] {...}

pred NewBid [e,e' : Ebay, c : Client, p : Product] {...}

assert NewBidOk {...}
```

   (a) Qual o objectivo dos invariantes `A`, `B`, e `C`?

   (b) Especifique o invariante `NoEqualBids`, cujo objectivo é impedir que haja dois bids de igual valor no mesmo leilão.

   (c) Especifique o predicado `IsCurrentWinner`, que deverá ser verdadeiro se o cliente `c` tem um bid vencedor num leilão para o produto `p`.

(d) Defina a asserção `NewBidOk`, cujo objectivo é garantir que a operação `NewBid` está correctamente especificada: deve ser garantida a preservação do invariante e que após a sua execução com sucesso o cliente `c` tem um bid vencedor num leilão para o produto `p`.

(e) Especifique a operação `NewBid` por forma a satisfazer a asserção `NewBidOk`.

2. Considere o seguinte modelo do sistema de informação do mestrado de informática:

```
sig Aluno {}
sig Grupo {
  membros : some Aluno
}
abstract sig UCE {}
one sig MFES, CSSI, SD, EA, ... extends UCE {}
sig Nota {}
sig MI {
  inscritos : UCE -> Aluno,
  grupos : UCE -> Grupo,
  notas : UCE -> Aluno -> Nota
}

pred A [m : MI] {
  all a : Aluno | #(m.inscritos.a) < 3
}
pred B [m : MI] {
  all u : UCE | m.notas[u].Nota in m.inscritos[u]
  all u : UCE | m.grupos[u].membros in m.inscritos[u]
}
pred C [m : MI] {
  all u : UCE, a : m.inscritos[u] | lone m.notas[u][a]
}
pred D [m : MI] {
  all u : UCE, a : m.inscritos[u] | one (m.grupos[u] & membros.a)
}
pred NotasGrupos [m : MI] { ... }

pred Inv [m : MI] {
  A[m] && B[m] && C[m] && D[m] && NotasGrupos[m]
}

pred LancaNota [m,m' : MI, u : UCE, a : Aluno, n : Nota] {
  // PRE
  a in m.inscritos[u]
  // POS
  m'.inscritos = m.inscritos
  m'.grupos = m.grupos
  m'.notas = m.notas + (u -> a -> n)
}
```

(a) Qual o objectivo dos invariantes `A`, `B`, `C`, e `D`?

(b) Especifique o invariante `NotasGrupos`, cujo objectivo é garantir que todos os elementos do mesmo grupo têm a mesma nota.

(c) Defina asserções e predicados para verificar a correcção e consistência da operação `LancaNota` em relação ao invariante definido.

(d) Corrija a especificação da operação `LancaNota` por forma a garantir a sua correcção e consistência.

3. Considere a seguinte especificação incompleta de um telemóvel. Para além da agenda telefónica, onde a cada nome estão associados vários números, o modelo guarda o conjunto de chamadas efectuadas e a hora actual.

```
open util/ordering[Hora]

sig Nome, Numero, Hora {}
sig Chamada {
  numero : Numero,
  hora : Hora
}
sig Telefone {
  agenda : Nome -> set Numero,
  chamadas : set Chamada,
  relogio : Hora
}

pred inv [t : Telefone] { ... }

pred novoNumero [t,t' : Telefone, n : Nome, i : Numero] { ... }

pred apagaNome [t,t' : Telefone, n : Nome] { ... }

pred chamar [t,t' : Telefone, n : Nome] { ... }

pred limparAntiga [t,t' : Telefone] { ... }
```

(a) Complete a definição do invariante do telefone por forma a satisfazer os seguintes requisitos:

- Um número não pode pertencer a duas pessoas diferentes.
- Todos os números chamados fazem parte da agenda.
- Não podem existir chamdas simultâneas.
- Todas as chamadas foram feitas antes da hora actual.
- Só é registada a última chamada efectuada para cada número.

(b) Complete a definição das seguintes operações (incluindo as respectivas pré-condições) e verifique a sua correcção em relação ao invariante:

- `novoNumero`: acrescentar um número à agenda.
- `apagaNome`: eliminar um nome da agenda, apagando todos os números que lhe estão associados.
- `chamar`: efectuar uma chamada para uma determinada pessoa - a chamada deve ficar registada com a hora actual, e o relogio deve avançar uma unidade em consequência da chamada.
- `limparAntiga`: eliminar o registo da chamada efectuada à mais tempo.

4. Considere o seguinte modelo de um gestor de memória. A relação `free` contém o conjunto de endereços livres. A relação `blocks` representa os blocos alocados: um bloco é essencialmente uma sequência de endereços, apontado pelo primeiro deles.

```
open util/ordering[Addr]
open util/integer
```

```
sig Addr {}
sig Memory {
   free : set Addr,
   blocks : Addr -> Addr
}

pred Inv [m : Memory] { ... }

pred Free [m, m' : Memory, a : Addr] { ... }

pred Alloc [m, m' : Memory, n : Int, a' : Addr] { ... }
```

(a) Como poderia especificar no predicado `Inv` os seguintes invariantes?

i. Não existem endereços simultanemaente livres e ocupados.

ii. Um endereço não pode estar alocado em dois blocos diferentes.

iii. O apontador para um bloco faz parte dos endereços alocados nesse bloco.

iv. Os endereços de um bloco são sequenciais e começam no apontador para o mesmo.

(b) Especifique a operação `Free` que liberta os endereços do bloco apontado por `a`.

(c) Como pode verificar a correcção e consistência desta operação?

(d) Especifique a operação `Alloc` que aloca `n` endereços e retorna o apontador `a'` para o novo bloco.

5. Considere o seguinte desafio. Numa colónia de camaleões cada um dos camaleões pode, em cada momento, ter uma das seguintes cores: vermelho, azul ou verde. Sempre que dois camaleões de cores diferentes se encontram, ambos mudam para a cor que nenhum deles tinha. Caso contrário não mudam de cor. O objectivo do desafio é, dada uma colónia inicial, descobrir qual a sequência de encontros que leva a que a colónia fique toda da mesma cor. Considere o seguinte modelo Alloy (incompleto) que pretende especificar este desafio.

```
open util/ordering[Colonia]

sig Camaleao {}
abstract sig Cor {}
one sig Vermelho, Azul, Verde extends Cor {}
sig Colonia {
  cor : Camaleao -> Cor,
  vizinho : Camaleao -> Camaleao
}

pred inv[c : Colonia] { ... }

fact { all c : Colonia | inv[c] }

pred muda[c, c' : Colonia, x : Camaleao] { ... }

pred nao_muda[c, c' : Colonia, x : Camaleao] { ... }

pred uniforme[c : Colonia] { ... }

fact {
  not uniforme[first]
  all c : Colonia, c' : c.next, x : Camaleao {
```

```
    muda[c,c',x] or nao_muda[c,c',x]
  }
}

run { some c : Colonia | uniforme[c] } for 3 but exactly 6 Camaleao
```

O módulo `ordering` é utilizado para especificar a evolução da colónia ao longo do tempo: um átomo do tipo `Colonia` representa o estado da colónia num determinado instante do tempo (e não uma colónia diferente). A relação `cor` representa a cor de cada camaleão em cada instante. A relação `vizinho` representa a relaxão de proximidade entre camaleões. O comando `run` procura sequências de encontros que levem a uma colónia uniforme.

(a) Como poderia especificar no predicado `inv` os seguintes invariantes?

   i. Em cada instante cada camaleão possui exactamente uma cor.

   ii. Nunca se encontram mais do que dois camaleões.

   iii. Um camaleão não se encontra com ele próprio.

   iv. Os encontros são recíprocos.

(b) Especifique o predicado `uniforme` que testa se todos os camaleões tem a mesma cor.

(c) Especifique a operação `nao_muda` que determina quando e como evolui um camaleao que não muda de cor entre dois instantes de tempo consecutivos.

(d) Especifique a operação `muda` que determina quando e como evolui um camaleao que muda de cor entre dois instantes de tempo consecutivos.

(e) É possível verificar usando o Alloy Analyzer que uma determinada colónia nunca converge para uma população uniforme? Justifique.