Exercícios sobre Alloy

Alcino Cunha

9 de Novembro de 2017

1. Considere o seguinte modelo do sistema de informação do mestrado em engenharia informática:

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

   (a) Especifique os seguintes invariantes:
   - Cada aluno só pode estar inscrito no máximo em duas UCEs.
   - Todos os alunos dos grupos de uma UCE estão inscritos nessa UCE.
   - Apenas os alunos inscritos têm (no máximo uma) nota em cada UCE.
   - Cada aluno inscrito pertence apenas a um grupo em cada UCE.
   - Todos os elementos de um grupo que já tem nota lançada têm a mesma nota.

   (b) Refine o modelo por forma a capturar a mutabilidade das relações `inscritos`, `grupos` e `notas`. Utilize o *local state idiom*.

   (c) Especifique a operação `LancaNota` que lanca a nota de um aluno de uma UCE, por forma a garantir a sua consistência e a preservação dos invariantes.

2. 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 Numero {}
sig Hora {}
one sig Actual in Hora {}
sig Nome {
  agenda : some Numero
}
sig Chamada {
  numero : one Numero,
  hora : one Hora
}
```

(a) Especifique os seguintes invariantes:

- 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.

(b) Refine o modelo por forma a capturar a mutabilidade das relações `Nome`, `agenda`, `Actual` e `Chamada`. Utilize o *local state idiom*.

(c) Especifique as seguintes operações, garantindo a sua consistência e garantindo que o relógio avança:

- `novo`: acrescentar um número à agenda.
- `apaga`: 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.

(d) Refine o modelo por forma a garantir que as operações preservam os invariantes especificados (continuando a ser consistentes).

(e) Especifique as mesmas operações usando o *event idiom*: tente tirar partido da hierarquia para factorizar os propriedades comuns.

(f) Usando o *trace idiom* especifique sequências de eventos válidos partindo de um telefone sem qualquer informação. Especifique a seguinte propriedade sobre traços: só é registada a última chamada efectuada para cada número. Altere a especificação das suas operações por forma a garantir esta propriedade.

3. Considere o seguinte modelo Alloy de um sistema de gestão de CV académico.

```
abstract sig UID {}
abstract sig Source {}
sig Publication {
  source : one Source,
  uids : some UID
}
sig Author extends Source {
  profile : set Publication,
  visible : set Publication
}
sig External extends Source {}

fun same[a : Author, p : Publication] : set Publication {
  let aux = a.profile <: uids | p.*(aux.~aux)
}
```

Neste sistema um autor pode ter no seu perfil várias publicações científicas, podendo escolher quais estão visíveis para o público em geral. Um publicação pode ser adicionado ao perfil pelo próprio autor ou por uma fonte (`Source`) externa, por exemplo, uma editora. Para efeitos deste modelo, para cada publicação apenas serão registadas a fonte do respectivo registo e o conjunto de identificadores únicos (`UIDs`) que a caracterizam (nomeadamente, identificadores únicos reconhecidos internacionalmente, como, por exemplo, o DOI). É normal que, para a mesma publicação, existam vários registos da mesma originários de fontes diferentes. A função `same`, dado um autor e uma publicação, calcula o conjunto de publicações no perfil do autor que de facto representam a mesma publicação (porque partilham directa ou indirectamente identificadores únicos).

(a) Especifique os seguintes invariantes sobre este modelo:

- As publicações visíveis tem que pertencer ao perfil do utilizador.
- No perfil de um utilizador apenas podem existir publicações cuja fonte é externa ou ele próprio.
- Uma fonte não pode adicionar ao perfil de um utilizador duas publicações que partilhem directamente identificadores únicos.
- No perfil de um utilizador não podem estar visíveis duas versões da mesma publicação.

(b) Altere o modelo por forma a permitir que as relações `profile`, `visible` e `Author` possam ser mutáveis. Modifique os invariantes por forma a acomodar esta alteração e especifique outros que julgue necessários.

(c) Usando predicados especifique as seguintes operações, garantindo que preservam os invariantes e que são consistentes (indique também como pode verificar estas propriedades):

- Remover um autor do sistema.
- Adicionar uma nova publicação ao perfil de um autor.

(d) Verifique se a seguinte propriedade de *safety* é ou não verdade. Se não o for indique como pode usar o idioma de traços para saber qual a sequência de operações que leva a um estado onde é inválida.

- Não é possível existirem duas versões da mesma publicação num perfil de utilizador publicados pela mesma fonte.

(e) Utilizando as relações deste modelo indique duas expressões que contenham, respectivamente, um erro de tipos detectável com *bounding types* e um erro de tipos detectável apenas com *relevance types*.

4. Considere o seguinte modelo de um gestor de memória. O conjunto `Free` contém o conjunto de endereços livres. A relação `block` indica qual o bloco a que um endereço está alocado. A relação `addr` captura os apontadores para os blocos.

```
open util/ordering[Addr]

sig Addr {
  block : lone Block
}
sig Free in Addr {}
sig Block {
  addr : one Addr
}
```

Na Figura 1 temos um exemplo de uma configuração válida da memória. Neste caso existem dois blocos alocados, `Block0` (ocupando os endereços `Addr4`, `Addr5` e `Addr6`) e `Block1` (ocupando os endereços `Addr0` e `Addr1`). O apontador para o bloco `Block0` é o endereço `Addr4` e o apontador para o bloco `Block1` é o endereço `Addr0`. Os endereços `Addr2` e `Addr3` estão livres.

(a) Especifique os seguintes invariantes:

- Não existem endereços simultanemaente livres e ocupados.
- O apontador para um bloco faz parte dos endereços alocados nesse bloco.
- Os endereços de um bloco são sequenciais.
- O apontador para um bloco é o primeiro dos endereços que lhe estão alocados.

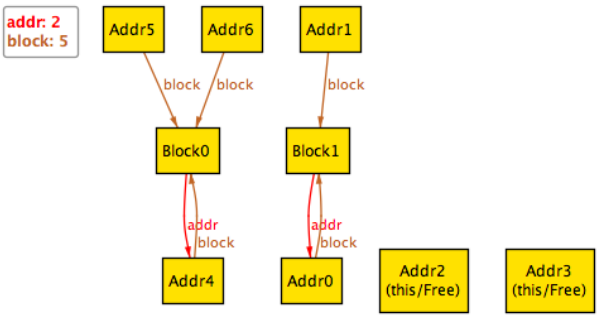

Figura 1: Exemplo de uma configuração válida da memória

(b) Verifique que estes invariantes garantem que:

- Os blocos não podem ser vazios.

(c) Refine o modelo por forma a capturar a mutabilidade das relações `Free`, `Block`, `block` e `addr`. Utilize o *global state idiom*.

(d) Especifique as seguintes operações, garantindo a sua consistência:

- `free`: dado um endereço liberta o bloco por ele apontado.
- `alloc`: dado um inteiro estritamente positivo aloca um bloco com esse tamanho e devolve o endereço para o mesmo.

(e) Refine o modelo por forma a garantir que as operações preservam os invariantes especificados (continuando a ser consistentes).

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

```
open util/ordering[Oferta]

sig Produto, Oferta {}
sig Leilao {
  produto : Produto
}
sig Cliente {
  leiloes : set Leilao,
  ofertas : Leilao -> Oferta
}
```

(a) Especifique os seguintes invariantes:

- Cada leilão pertence a um cliente.
- Um cliente não pode licitar produtos que esteja a leiloar.
- Não pode haver duas ofertas de igual valor no mesmo leilão.

(b) Verifique que estes invariantes garantem que:

- Um cliente não licita nos seus próprios leilões.

(c) Refine o modelo por forma a capturar a mutabilidade das relações `Leilao`, `leiloes` e `ofertas`. Utilize o *local state idiom*.

(d) Especifique a operação `Leiloar` que para um cliente `c` cria um leilão para o produto `p`. Verifique a sua consistência.

(e) Especifique o predicado `Vencedor` que testa se um cliente `c` tem uma oferta vencedora num leilão de um produto `p`.

(f) Especifique a operação `Licitar` que para um cliente `c` faz uma uma oferta vencedora num produto `p`. Verifique que essa operação é consistente e que realmente garante que `c` é `Vencedor` para `p`.

(g) Refine o modelo por forma a garantir que as operações preservam os invariantes especificados (continuando a ser consistentes).

6. Considere o seguinte modelo Alloy de um programa Java.

```
sig Class {
  super : lone Class,
  vars : set Var
}
one sig Object extends Class {}
sig Var {
  name : one Name,
  type : one Class
}
sig Name {}
```

Um programa é constituído por classes, declarando cada classe um conjunto de variáveis. Cada variável tem um nome e um tipo, que por sua vez é uma classe. A classe `Object` é uma classe especial que existe sempre. Uma classe pode ainda extender outra classe, sendo esta designada por super classe da primeira. Uma classe herda as variáveis declaradas na sua super classe, pelo que esta relação entre classes é normalmente conhecida por relação de herança.

(a) Especifique os seguintes invariantes sobre este modelo:

- A class `Object` não tem variáveis declaradas.
- Não é possível numa classe declarar duas variáveis com o mesmo nome.
- Todas as classes, com excepção de `Object`, têm uma super classe.
- A relação de herança entre classes é acíclica.

(b) Altere o modelo por forma a permitir que as relações `Class`, `super` e `vars` possam ser mutáveis. Modifique os invariantes por forma a acomodar esta alteração e especifique outros que julgue necessários.

(c) Usando predicados especifique as seguintes operações, garantindo que preservam os invariantes e que são consistentes (indique também como pode verificar estas propriedades):

- Adicionar uma variável a uma classe.
- Remover uma classe.

(d) Verifique se a seguinte propriedade de *safety* sobre traços é ou não verdade, assumindo que apenas as duas operações anteriores podem executar a partir de um estado inicial qualquer que satisfaz o invariante acima especificado.

- So é possível executar a operação de adicionar uma variável a uma classe se a operação de remover essa classe não tiver sido executada antes.

(e) Justifique porque é que esta é uma propriedade de *safety* (e não de *liveness*), e indique um exemplo de uma propriedade de *liveness* para este modelo.

7. Especifique o problema da coloração de grafos em Alloy e utilize o Analyzer para descobrir uma coloração mínima do grafo de Peterson. Mais informação em:

`http://en.wikipedia.org/wiki/Graph_coloring`

8. Todos os artigos submetidos a uma conferência científica são revistos por um ou mais elementos da respectiva comissão de programa, que eventualmente decide quais deles devem ser aceites para apresentação e inclusão nas actas. Cada revisão inclui uma apreciação objectiva da qualidade do artigo sobre a forma de uma classificação (dentro de uma gama fixa de notas). O seguinte modelo Alloy especifica um *snapshot* deste processo.

```
open util/ordering[Nota]

sig Pessoa {}
some sig Comissao in Pessoa {}
sig Nota {}
sig Artigo {
  autores : some Pessoa,
  nota : Pessoa -> lone Nota
}
sig Submetido, Aceite in Artigo {}
```

   (a) Especifique os seguintes invariantes:
      - As revisões só podem ser feitas por membros da comissão de programa a artigos submetidos.
      - Um artigo não pode ser revisto pelos seus autores.
      - Todos os artigos aceites tem que ter pelo menos uma revisão.
      - Todos os artigos com uma nota máxima são automaticamente aceites.

   (b) Verifique que estes invariantes garantem que:
      - Só são aceites artigos que foram submetidos.

   (c) Refine o modelo por forma a capturar a mutabilidade das relações `Submetido`, `Aceite` e `nota`. Utilize o *global state idiom*.

   (d) Especifique as seguintes operações, garantindo a sua consistência:
      - `submeter`: submeter um artigo.
      - `aceitar`: aceitar um artigo.
      - `rever`: classificar um artigo submetido.

   (e) Refine o modelo por forma a garantir que as operações preservam os invariantes especificados (continuando a ser consistentes).

   (f) Usando o *trace idiom* especifique sequências de eventos válidos. Especifique a seguinte propriedade sobre traços: não é possível rever as classificações de um artigo para valores mais baixos. Altere a especificação das suas operações por forma a garantir esta propriedade.

9. O número de Erdös mede a "distância", em termos de co-autoria de artigos, de um investigador ao prolífico matemático Paul Erdös. O número de Erdös de Paul Erdös é 0, o número de Erdös de todos os seus co-autores é 1, o dos co-autores dos seus co-autores é 2, e assim sucessivamente. Se não houver "caminho" entre uma pessoa e Paul Erdös o seu número de Erdös é indefinido. Mais informações sobre este número podem ser encontradas na respectiva página da Wikipedia:

   `http://en.wikipedia.org/wiki/Erdos_number`

   O modelo do exercício anterior pode ser adaptado por forma a especificar este número. Basta modificar a assinatura `Pessoa` e acrescentar Paul Erdös:

```
open util/natural

sig Pessoa {
  erdos: lone Natural
}
one sig Erdos extends Pessoa {}
```

Defina um facto que garante que o número de Erdös de cada pessoa é correcto, de acordo com a especificação informal dada acima.

10. Uma empresa fabrica componentes numa linha de montagem organizada em fases de fabrico sucessivas. Um componente é fabricado numa das fases de fabrico usando materiais base e/ou sub-componentes fabricados previamente. Cada fase de fabrico deve ser suportada por máquinas que fazem a montagem dos componentes lá fabricados. O seguinte modelo Alloy (incompleto) especifica formalmente uma visão estática deste processo.

```
open util/ordering[Fase]

sig Fase {}

abstract sig Produto {}

sig Componente extends Produto {
  partes : set Produto,
  fase : one Fase
}

sig Material extends Produto {}

sig Maquina {
  fase : one Fase
}
```

(a) Especifique invariantes que garantam as seguintes propriedades:
- Um componente não pode ser fabricado a partir do nada.
- Um componente não pode ser um dos seus sub-componentes.
- Todos os sub-componentes de um componente tem que ser fabricados em fases prévias.
- Todas as fases envolvidas no fabrico de algum componente tem que ser suportadas por máquinas.

(b) Altere o modelo por forma a permitir a modificação do catálogo de produtos fabricados, ou seja, permitir que as relações `Componente`, `partes`, e `fase` (do componente) sejam mutáveis, continuando a garantir os invariantes anteriores e potenciais restrições de integridade referencial. Utilize o *local-state idiom*, utilizando a assinatura `State` para denotar o estado.

(c) Especifique uma operação que permita acrescentar um novo sub-componente ou material (um `Produto`) às partes necessárias para fabricar um componente, garantindo que essa operação preserva os invariantes anteriores.

(d) Considere agora que existe um stock de materiais especificado da seguinte forma:

```
sig Material extends Produto {
  stock : Int -> State
}
```

```
fact {
  all s : State, m : Material {
    one m.(stock.s) and gte[m.(stock.s),0]
  }
}
```

Especifique um predicado que teste se um componente pode ser fabricado num determinado estado, ou seja, existe material suficiente em stock para o fabricar (incluindo todos os sub-componentes).

(e) Usando o modelo apresentado no início do teste, apresente três exemplos de expressões que apresentem erros devido a *bounding types* vazios, *relevance types* vazios, e ambiguidade, respectivamente. Justifique formalmente a sua resposta.

11. 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.

```
sig Camaleao {
  cor : one Cor,
  encontra : lone Camaleao
}
abstract sig Cor {}
one sig Vermelho, Azul, Verde extends Cor {}
```

(a) Especifique os seguintes invariantes:

- Um camaleao não se encontra com ele próprio.
- Os encontros são recíprocos.

(b) Refine o modelo por forma a capturar a mutabilidade das relações `cor` e `encontra`. Utilize o *local state idiom*.

(c) Utilizando o *trace idiom* especifique as dinâmica da colonónia.

(d) Utilize o Alloy Analyzer para determinar a sequência de encontros necessários para levar a que uma dada colónia fique uniforme.

(e) Tente inferir qual a característica geral das colónias com potencial de ficarem uniformes.

12. Considere o seguinte modelo dinâmico, com duas operações `X` e `Y` modeladas com eventos e *frame conditions* ao estilo de Reiter.

```
sig State {}

sig A {
  r : set State, s : set State, t : set State
}

abstract sig Event {
  pre: State, pos: State,
  a : A
} {
  r.pos = r.pre + a
  s.pos != s.pre implies this in X
  t.pos != t.pre implies this in Y
}
```

```
sig X extends Event {} {
  s.pos = s.pre + a
}

sig Y extends Event {} {
  t.pos = t.pre + a
}
```

Defina um modelo equivalente especificando as operações usando predicados.