Gerenciamento de árvores genealógicas no sistema *Alloy*

Bolivar Pereira
Eduardo Cardoso
Gabriel Brunichaki
Gabriel Weich

06 de Julho de 2020

Resumo

Este relatório tem por objetivo descrever uma solução para o problema proposto para o segundo trabalho da disciplina de Métodos Formais do sétimo nível do curso de Bacharelado em Engenharia de Software da Pontifícia Universidade Católica do Rio Grande do Sul, que consiste na construção de especificações e verificações formais no sistema *Alloy*.

Introdução

Dentro do escopo da disciplina de Métodos Formais, o problema proposto no segundo trabalho consiste em implementar especificações e verificações formais para um sistema de gerenciamento de árvores genealógicas utilizando a linguagem *Alloy*.

Alloy é uma linguagem *open source* de especificação de modelos que tem como base a Teoria de Conjuntos. Os modelos podem ser vistos como definições de classes e seus atributos em uma linguagem orientada a objetos. Além disso, *Alloy* também possui um aplicativo para interpretação e análise de modelos, que permite explorar e testar os modelos, buscando contraexemplos que demonstrem problemas na especificação.

O sistema possui três funcionalidade principais, sendo elas:

- Manter um modelo de dados correto para uma árvore genealógica com informações de relações de paternidade/maternidade entre as pessoas, relações de casamento e informações de vivos e mortos.
- Permitir cálculos de relações familiares derivadas como irmãos, avós, netos, tios, primos, descentes de todos os níveis.
- Operações de adição de pessoas à árvore genealógica, operação de casamento entre pessoas, operação de divórcio entre pessoas.

1 Assinaturas

Iniciando nossa solução, temos "Tempo"como a primeira assinatura, que representa um determinado tempo no sistema e é ordenada de forma ascendente. A assinatura "Tempo"é utilizada pela extensão *util/ordering* para gerenciar a sequência de estados de forma a permitir a modelagem dos aspectos dinâmicos do modelo.

Temos também uma assinatura abstrata "Pessoa"que possui os atributos "filhos", "conjuge"e "vivo", onde "filhos"representa os filhos de uma pessoa mapeados como um conjunto de pessoas em um dado momento no tempo. "Conjuge"representa o cônjuge daquela pessoa em um determinado tempo. O atributo "vivo"define se a pessoa está viva em um determinado tempo.

Finalizando as assinaturas, criamos também "Homem"e "Mulher", que são assinaturas que estendem a assinatura "Pessoa". As implementações das assinaturas podem ser vistas a seguir:

```
open util/ordering [Tempo] as T

sig Tempo {}

abstract sig Pessoa {
  filhos: Pessoa set -> Tempo,
  conjuge: Pessoa lone -> Tempo,
  vivo: set Tempo
}

sig Homem, Mulher extends Pessoa {}
```

2 Funções

Fora a relação de filhos e cônjuge, todos os demais graus de parentesco de uma pessoa são calculados através de funções no *Alloy*. Sendo assim, criamos funções para pais, netos, descendentes, avôs, irmãos, tios e primos, conforme mostra a imagem abaixo:

```
fun pais [t: Tempo]: Pessoa -> Pessoa {
  ~ (filhos.t)
}

fun netos [t: Tempo]: Pessoa -> Pessoa {
  filhos.t.filhos.t
}

fun descendentes [t: Tempo]: Pessoa -> Pessoa {
  ^ (filhos.t)
}

fun avos [t: Tempo]: Pessoa -> Pessoa {
  ~ (filhos.t.filhos.t)
}

fun irmaos [t: Tempo]: Pessoa -> Pessoa {
  pais[t].filhos.t
}

fun tios [t: Tempo]: Pessoa -> Pessoa {
  pais[t] . (irmaos[t])
}
```

```
fun primos [t: Tempo]: Pessoa -> Pessoa {
  tios[t].filhos.t
}
```

Segue a descrição das funções:

- Pais: Os pais são definidos pelo inverso da relação de filhos em um determinado ponto do tempo.
- Netos: Os netos são obtidos através dos filhos dos filhos de uma pessoa em um determinado ponto do tempo.
- Descendentes: Os descendentes são obtidos pela aplicação do fecho transitivo não-reflexivo da relação de filhos.
- Avós: inverso da relação de netos.
- Irmãos: filhos dos pais em um determinado ponto do tempo.
- Tios: Os tios são obtidos através dos irmãos dos pais em um determinado ponto do tempo.
- Primos: filhos dos tios em um determinado ponto do tempo.

3 Predicados

Na linguagem *Alloy* os predicados permitem definir uma restrição reutilizável que aplicada a uma determinada entrada avalia se a mesma satisfaz ou não a restrição estabelecida.

Os predicados "NaoMudaFilhos", "NaoMudaConjuges"e "NaoMudaVivo"servem para assegurar que, respectivamente, os filhos, cônjuges e o estado de vida de um conjunto de pessoas não mudam, dados dois pontos diferentes no tempo.

```
pred NaoMudaFilhos [ps: set Pessoa, t, t': Tempo] {
  all p: ps | p.filhos.t' = p.filhos.t
}

pred NaoMudaConjuges [ps: set Pessoa, t, t': Tempo] {
  all p: ps | p.conjuge.t' = p.conjuge.t
}

pred NaoMudaVivo [ps: set Pessoa, t, t': Tempo] {
  all p: ps | p in vivo.t' iff p in vivo.t
}
```

Os predicados "Nascer", "Casar"e "Divorciar"definem as operações de adição de pessoas à árvore genealógica, casamento entre duas pessoas e divórcio entre duas pessoas, respectivamente.

```
pred Nascer [p: Pessoa, m: Homem, w: Mulher, t,t': Tempo] {
  m+w in vivo.t
  p !in vivo.t

  m.conjuge.t = w
  w.conjuge.t = m

  vivo.t' = vivo.t + p
```

```
  m.filhos.t' = m.filhos.t + p
  w.filhos.t' = w.filhos.t + p

  NaoMudaFilhos[Pessoa - m - w, t, t']
  NaoMudaConjuges[Pessoa, t, t']
  NaoMudaVivo[Pessoa - p, t, t']
}

pred Casar [m: Homem, w: Mulher, t, t': Tempo] {
  m+w in vivo.t
  m.conjuge.t = none
  w.conjuge.t = none

  m.conjuge.t' = w
  w.conjuge.t' = m

  NaoMudaFilhos[Pessoa, t, t']
  NaoMudaConjuges[Pessoa - w - m, t, t']
  NaoMudaVivo[Pessoa, t, t']
}

pred Divorciar [m: Homem, w: Mulher, t, t': Tempo] {
  m+w in vivo.t

  m.conjuge.t = w
  w.conjuge.t = m

  m.conjuge.t' = none
  w.conjuge.t' = none

  NaoMudaFilhos[Pessoa, t, t']
  NaoMudaConjuges[Pessoa - w - m, t, t']
  NaoMudaVivo[Pessoa, t, t']
}
```

Segue a descrição dos predicados:

- Nascer: Torna uma pessoa "p"viva e adiciona ela aos filhos de um casal.
  - Pré-condições: Homem e Mulher estejam vivos e sejam casados e a pessoa "p"ainda não esteja viva.
  - Pós-condições: A pessoa "p"estará viva e fará parte do conjunto de filhos do Homem e da Mulher.
- Casar: Cria um vínculo entre um Homem e uma Mulher.
  - Pré-condições: Homem e Mulher estejam vivos e ainda não tenham cônjuges.
  - Pós-condições: Homem e Mulher tornam-se cônjuges um do outro.
- Divorciar: Desvincula a relação entre um Homem e uma Mulher.
  - Pré-condições: Homem e Mulher estejam vivos e sejam casados um com o outro.

– Pós-condições: Homem e Mulher não possuem mais cônjuges.

Os predicados "NaoMudaFilhos", "NaoMudaConjuges"e "NaoMudaVivo"foram utilizados para assegurar que as alterações ocorram apenas para as pessoas especificadas nos parâmetros do predicado, todo o resto mantêm-se inalterado.

4 Fatos

Foi criada também uma série de fatos para impedir que o modelo se comportasse em situações impossíveis na realidade modelada, como o de uma pessoa ser casada consigo mesma. Os fatos criados se encontram abaixo:

```
fact naoPodeCasarConsigoMesmo {
  all t: Tempo { no p:Pessoa | p.conjuge.t = p }
}

fact naoPodeCasarComDescendente {
  all t: Tempo { no p:Pessoa | p.conjuge.t in p.^(filhos.t) iff p.conjuge.t
    ↪ != none }
}

fact naoPodeSerFilhoDeDescendente {
  all t: Tempo { no p:Pessoa | p in p.^(filhos.t) }
}

fact casamentoReciproco {
  all t: Tempo {
    all p: Pessoa | p.conjuge.t.conjuge.t = p iff p.conjuge.t != none
  }
}

fact casamentoHomemMulher {
  all t: Tempo {
    all p: Pessoa | let c = p.conjuge |
      (p in Homem implies c.t in Mulher) and
      (p in Mulher implies c.t in Homem)
    }
}
```

Segue a descrição dos fatos:

- naoPodeCasarConsigoMesmo: Assegura que o cônjuge de uma pessoa não é ela mesma.
- naoPodeCasarComDescendente: Assegura que se uma pessoa estiver casada seu cônjuge não seja seu descendente.
- naoPodeSerFilhoDeDescendente: Assegura que uma pessoa não é filha de um descendente seu.
- casamentoReciproco: Assegura que em um casal o cônjuge de um é o cônjuge do outro.
- casamentoHomemMulher: Assegura que um cônjuge homem tenha um cônjuge mulher vice-versa.

5 Asserções

As asserções são utilizadas para definir propriedades a serem verificadas. O comando "check"é responsável por verificar uma asserção buscando um contraexemplo para a mesma em um escopo limitado.

A fim de verificar as funções de parentesco criadas, criamos algumas asserções que estão listadas abaixo:

```
assert assertNetos {all t:Tempo {all p: netos[t] | p in avos[t][p].filhos.t.
    ↪ filhos.t}}
assert assertPrimos {all t:Tempo {all p: primos[t] | p in tios[t][p].filhos.t
    ↪ }}
assert assertIrmaos {all t:Tempo {all p: irmaos[t] | p in pais[t][p].filhos.t
    ↪ }}
assert assertIrmaosDescendentesPai {all t: Tempo {all p: irmaos[t] | p in
    ↪ pais[t][p].(descendentes[t])}}

check assertNetos for 8
check assertIrmaos for 8
check assertPrimos for 8
check assertIrmaosDescendentesPai for 8
```

Segue a descrição das asserções:

- assertNetos: Verifica para o conjunto dos netos se estão no conjunto dos filhos dos filhos dos avós.
- assertPrimos: Verifica para o conjunto dos primos se estão no conjunto dos filhos dos tios.
- assertIrmaos: Verifica para o conjunto dos irmãos se estão no conjunto dos filhos dos pais.
- assertIrmaosDescendentesPai: Verifica para o conjunto de irmãos se estão incluídos nos descendentes dos pais.

Todas as asserções são verificadas para o conjunto total dos tempos do sistema.

6 Conclusão

Por fim, este relatório teve como base a demonstração de como implementamos especificações e verificações formais para um sistema de gerenciamento de árvores genealógicas utilizando a linguagem de especificação de modelos *Alloy*.

Ao decorrer do desenvolvimento da solução proposta, desenvolvida na ferramenta em questão, foi possível realizar a verificação da estrutura e do comportamento de predicados, funções e fatos construídos. Desta forma, foi possível cobrir o problema em sua totalidade e especificidade. Além disso, demonstramos indícios, através de exemplos de asserções verificáveis no *Alloy*, da correção da modelagem apresentada.