

Especificação e Modelação
Relatório

Meta-Modelação

Mestrado
em
Engenharia Informática

Realizado por

27754 **Vitor Enes Duarte**



Departamento de Informática
UNIVERSIDADE DO MINHO
Braga, Portugal

29 de Dezembro de 2015

Conteúdo

| | | |
|----------|--|----------|
| 1 | Meta-Modelação em Alloy | 1 |
| 2 | Módulo util | 2 |
| 3 | Módulo model | 3 |
| 3.1 | Model | 3 |
| 3.2 | Name e Signature | 3 |
| 3.3 | Relation | 4 |
| 3.4 | ModelInstance, Atom e RelationInstance | 6 |
| 3.5 | Outros predicados | 7 |
| 4 | Módulo main | 8 |

1 Meta-Modelação em Alloy

Neste trabalho foi nos proposto usar o *Alloy* para especificar o próprio *Alloy*, focando-nos na parte estrutural. O resultado do trabalho pode ser encontrado em github.com/vitorennesduarte/meta-alloy.

No repositório podemos encontrar:

- **meta/util.als:** conjunto de funções auxiliares (talvez dispensável visto que acabou por *#conjunto = 1*)
- **meta/model.als:** assinaturas, os seus factos implícitos, e conjunto de funções e predicados sobre estas assinaturas
- **meta/main.als:** *run*'s e *check*'s
- **theme.thm:** tema usado ao longo do desenvolvimento do trabalho
- **report.pdf:** this

2 Módulo util

Neste módulo está apenas definida a função *range*:

```
module meta/util

open util/integer

fun range[min : Int, max : Int] : set Int {
  min.*next & *next.max
}
```

3 Módulo model

3.1 Model

O nosso modelo de um modelo em *Alloy* (*Model*) consiste num conjunto de *Signature* e num conjunto de *Relation*.

```
one sig Model {  
  signatures : set Signature,  
  relations : set Relation  
}
```

3.2 Name e Signature

A cada *Signature* está associado um nome.

```
sig Name {}  
  
sig Signature {  
  sigName : one Name  
}
```

Mas ao contrário do *Model*, as *Signature* têm factos implícitos (na verdade há um facto implícito no *Model*, é apenas *one*).

```
sig Signature {  
  sigName : one Name  
}{  
  all s : Signature | facts[s]  
}  
  
pred facts[s : Signature] {  
  s in Model.signatures  
  #(s.sigName).~sigName = 1  
}
```

O primeiro facto indica que uma *Signature* pertence ao conjunto de *Signature* do *Model*. O segundo garante que o nome da *Signature* é único.

3.3 Relation

As *Relation* foram modeladas da seguinte maneira:

```
sig Relation {  
  relName : one Name ,  
  relation : Int -> Signature  
}
```

Em *Haskell* podíamos definir estas duas relações como:

```
relName :: Relation -> Name  
relation :: Relation -> Int -> Signature
```

A primeira não apresenta nada de novo: é uma relação binária como todas as que vimos até agora. A segunda, uma relação ternária, foi a maneira escolhida para representar as relações do *Alloy*. A ideia inicial era representar estas relações como uma lista ligada. No entanto, essa escolha iria tornar mais difícil (se calhar impossível) a definição de certas funções/predicados (*e.g.* decidir se uma instância de uma relação é válida).

Esta opção traz ainda outra vantagem que só nos apercebemos mais tarde. Há um operador relacional do *Alloy* que tornou o desenvolvimento do trabalho muito agradável: [] - **box (join)**.

Podemos pensar na *relation* como uma relação que dada uma *Relation* nos dá um mapa que associa a cada *Int* (key) uma *Signature* (value). Isto é (usando o *Evaluator*):

```
M/Relation$0.relation  
  
{0->M/Signature$2, 1->M/Signature$2,  
 2->M/Signature$1}
```

Se, por exemplo em *Javascript*, quisesse saber qual o *domain* desta relação podia simplesmente na *Bash*: (**domain.js** definido mais à frente)

```
$ chmod u+x domain.js  
$ ./domain.js  
Signature2
```

```
#!/usr/bin/env node

function domain(relation) {
    return relation[0];
}

var Relation0 = {
    0: 'Signature2',
    1: 'Signature2',
    2: 'Signature1',
}

console.log(domain(Relation0));

domain.js
```

Acontece que em **Alloy**, graças ao operador **box join**, podemos fazer o mesmo (fixe):

```
fun keys[r : Relation] : set Int {
    (r.relation).Signature
}

fun lastKey[r : Relation] : Int {
    max[keys[r]]
}

fun arity[r : Relation] : Int {
    #(keys[r])
}

fun domain[r : Relation] : Signature {
    r.relation[0]
}

fun range[r : Relation] : Signature {
    r.relation[lastKey[r]]
}
```

Necessitamos ainda de definir alguns factos sobre cada *Relation*.

```
pred facts[r : Relation] {  
  // * a relation belongs to the model relations  
  r in Model.relations  
  
  // * min key is 0  
  let K = keys[r] | min[K] = 0  
  
  // * the keys are consecutive  
  let K = keys[r] | K = range[min[K], max[K]]  
  
  // * there's only one value for each key  
  all k : keys[r] | #(r.relation[k]) = 1  
}
```

3.4 ModelInstance, Atom e RelationInstance

Estas três assinaturas modelam as instâncias de *Model*, *Signature* e *Relation*, respectivamente.

3.5 Outros predicados

Neste módulo foram ainda definidos mais alguns predicados:

- **validJoin**: predicado que testa se um **dot join** é possível
- **join**: predicado que simula a operação **dot join**
- **isTransitive**: predicado que testa se uma *Relation* é transitiva
- **validRelationInstance**: predicado que dada uma *RelationInstance*, indica se esta é válida

```
pred join[s: Signature, r: Relation, r': Relation] {  
  // left join  
  validJoin[s, r]  
  arity[r] = plus[arity[r'], 1]  
  let K = keys[r] - 0 |  
    all k: K |  
      r'.relation =  
        r'.relation + (minus[k, 1] -> r.relation[k])  
}
```

4 Módulo main

É neste módulo que temos toda a acção. Alguns exemplos:

```
pred someTernaryRelation {
  // ...
  some r : Relation | arity[r] = 3
}

run someTernaryRelation for // ...

check twoLess {
  all r1, r2, r' : Relation, s : Signature |
    join[s, r1, r'] implies
      arity[r'] = minus[plus[arity[r1], arity[s]], 2]
  and join[r1, s, r'] implies
      arity[r'] = minus[plus[arity[r1], arity[s]], 2]
  and join[r1, r2, r'] implies
      arity[r'] = minus[plus[arity[r1], arity[r2]], 2]
}

check allTransitiveRelationsAreBinary {
  all r : Relation | isTransitive[r] implies arity[r] = 2
}
```

Na verdade twoLess era para se chamar *theArityOfAJoinIsAlwaysTwo-LessThanTheSumOfTheAritiesOfItsArguments*, mas não arranjei maneira de o encaixar na página.