FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO
Mestrado Integrado em Engenharia Informática e Computação

Métodos Formais em Eng.ª de Software

EXERCÍCIOS

1. Considerando as relações A e B indique o valor das expressões em Alloy:

A = {(a0,d0), (b0,a0), (b1,c1),(d0,d4),(d4,b1) }

a. {(a0),(b1)} <: A
   i. {(a0,d0),(b1,c1)}

b. {(a0)}.^A
   i. {(a0,d0),(a0,d4),(a0,b1),(a0,c1), (b0,a0),(b0,d0),(b0,d4),(b0,b1),(b0,c1), (b1,c1), (d0,d4),(d0,b1),(d0,c1), (d4,b1),(d4,c1)}
   ii. {(d0),(d4),(b1),(c1)}

2. Considerando as relações A, B, C e D, indique o valor das expressões em Alloy:

A = {(N1),(N2),(N3)}
C = {(N1,N1),(N1,N2),(N3,N2)}
D = {(X1,N2),(X1,N1),(X3,X1)}

a. C.A -> C[A]
   i. {(N1),(N3)} -> {(N1),(N2)}
   ii. {(N1,N1), (N1,N2), (N3,N1), (N3,N2)}

b. C ++ ~D
   i. C ++ {(N2,X1),(N1,X1),(X1,X3)}
   ii. {(N1,X1),(N3,N2),(N2,X1),(X1,X3)}

3. Considerando as relações A, B e C, indique o valor das expressões em Alloy:

A = {(a0,d0),(a1,c0),(d4,b1),(d0,d4)}
B = {(b0,a0),(b1,d4)}
C = {(c0,b0),(c1,b1),(d4,a0)}

a. C.B.A :> {(b1)}
   i. {(c1,b1)}

b. {(a0)}.^A
   i. {(d0),(d4),(b1)}

4. Considerando as relações A, B e C, indique o valor das expressões em Alloy:

A = {(a0,d0), (b0,a0), (b1,c1),(d0,d4),(d4,b1) }
B = {(b0,d4),(c1,a0)}

a. {(a0),(b1)} <: (A.~B)
   i. {(a0),(b1)} <: A.{(d4,b0),(a0,c1)}
   ii. {}

b. {(d0)}.^A
   i. {(d0)}.{(d0,d4),(d0,b1),(d0,c1),...}
   ii. {(d4),(b1),(c1)}

5. Considere a seguinte representação de um grafo em Alloy:

```
sig Point {}
sig Graph{
edge: Point -> some Point
}
```

Figura 1: Representação de um grafo em Alloy

a. Especifique uma função (*Insere*), em Alloy, que adicione uma aresta ao grafo e retorne esse grafo.

```
pred Insere[g:Graph,p1: Point, p2:Point,g':Graph]  {
   g'.edge = g.edge + p1->p2
}
// OR
fun Insere[g:Graph,p1,p2:Point]:Graph {
 {g':Graph | g'.edge = g.edge + p1->p2}
}
```

b. Se o grafo de *input* for bi-conexo, a função *Insere* (da alínea a.) garante que, após a sua execução, o grafo se mantém bi-conexo? Porquê? Em caso negativo, escreva uma pré-condição para a função *Insere* que seja suficiente para garantir a bi-conetividade do grafo resultante.

Não, porque posso adicionar um (ou dois) novo(s) vertice(s) ao grafo o que o tornaria não bi-conexo. A pré-condição poderia ser

```
(p1+p2) in (g.edge.Point+g.edge[Point])
```

c. Escreva uma função, em Alloy, que verifique a bi-conetividade de um grafo. Escreva também a verificação (*check*) em Alloy que lhe poderia testar a bi-conetividade antes e depois de aplicar a função *Insere* da alínea anterior.

```
pred biconnected[g:Graph]{
  all p:g.edge.Point+g.edge[Point] | connected[remove[g,p]]
}
fun remove[g:Graph, p:Point]: Graph{
 {g':Graph | g'.edge = ((Point-p)<: g.edge) :> (Point-p) }
}
pred connected[g:Graph] {
 {all p1,p2: g.edge.Point + g.edge[Point] | p1 in p2.^(g.edge + ~(g.edge))}
}
//OR
pred connected[g:Graph]{
 {all p1,p2:g.edge.Point+g.edge[Point] | p1->p2 in ^(g.edge + ~(g.edge))}
}
check {all g,g':Graph, p1:Point,p2:Point | biconnected[g] &&
Insere[g,p1,p2,g'] => biconnected[g']
}
```

d. Escreva uma função, em Alloy, que receba um grafo e retorne o seu grafo complementar (Figura 2).

```
pred Complementar[g,g':Graph]{
 g'.edge = [g.edge.Point+g.edge[Point] -> g.edge.Point+g.edge[Point]] –
[g.edge + ~(g.edge)]
}
// OR
pred Complementar[g,g':Graph]{
 g'.edge = ^(g.edge+~g.edge)-(g.edge+~(g.edge))
}
```

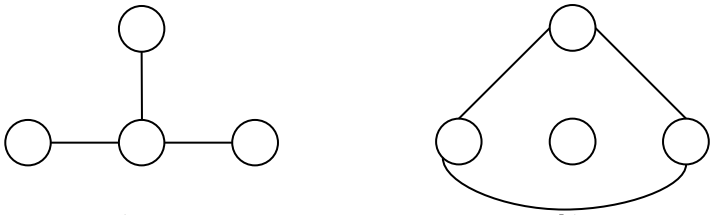

Figura 2: a) Grafo de entrada b) Grafo complementar de saída

6. Considere a seguinte representação de um polígono em Alloy (Fig. 3):

```
sig Point {}
sig Angulo {}
sig Poligono {
  vertices: set Point,
  lados: Point -> Point,
  angulo: Point -> one Angulo
}
```

Fig. 3: Representação de um grafo em Alloy

a. Especifique uma função, em Alloy, que verifique se o Polígono tem os ângulos todos iguais.

```
pred angiguais[p: Poligono] {
   one p.angulo[Point]
   // or
   #p.angulo[Point]
   }
 //OR
open util/boolean
fun AngIguais[p: Poligono] : Bool {
   { one p.angulo[Point] => True else False }
}
```

b. Especifique uma função, em Alloy, que verifique se um Polígono está bem construído, isto é, se dos lados e dos ângulos fazem parte apenas pontos que pertencem aos vértices desse Polígono.

```
pred wellDone[p:Poligono] {
   (p.angulo.Angulo + p.lados.Point + p.lados[Point]) in p.vertices
}
```

c. Especifique uma função, em Alloy, que retire um vértice do Polígono mantendo-o fechado.

```
fun removeVertice[p:Poligono, v:Point]:Poligono{
{p':Poligono | p'.vertices = p.vertices – v &&
    p'.angulo = p.angulo – v->p.angulo[v] &&
    p'.lados = p.lados ++ {(p.lados.v + p.lados[v])->(p.lados.v + p.lados[v])}
       – v->p.lados[v] –p.lados.v->v}
```

7. Considere a seguinte representação de uma árvore em Alloy (Fig.):

```
sig Point {}
sig Edge {
  top: one Point,
  right: lone Point,
  left: lone Point
}
sig Tree{
  root: Point,
  edges: set Edge
}
```

Figura 4: Representação de uma árvore em Alloy

a. Escreva uma função, em Alloy, que retorne o conjunto de folhas de uma árvore.

```
fun folhas2[t:Tree] : set Point{
  {f:Point| all e:Edge | t = edges.e && f = e.top && no (e.right+e.left)} +
   t.edges.right + t.edges.left - t.edges.top +
  {f:t.root | no t.edges }
}
```

b. Escreva uma função, em Alloy, que verifique se a raiz (*root*) da árvore é um nó (*Point*) da árvore.

```
pred raizInTree[t:Tree]{
  t.root in (t.edges.top + t.edges.right + t.edges.left)
}
```

c. Escreva uma função, em Alloy, que verifique se uma dada árvore é conexa.

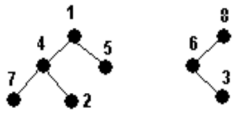

Figura 5: Exemplo de árvore desconexa

```
pred checkConexa[t: Tree] {
   all p: t.edges.top + t.edges.right + t.edges.left - t.root| caminho[t,p]
}

pred caminho[t:Tree, p: Point] {
   t.root->p in ^((~top).right + (~top).left)
}

check {all t:Tree |  checkConexa[t]}
```

8. O Klondike, mais conhecido por "solitário", é um jogo de 52 cartas (sem os Jokers) e apenas 1 jogador. O tabuleiro está dividido em três zonas: (1) duas pilhas, (2) quatro fundações, e (3) sete filas. O objectivo do jogo consiste em preencher as 4 fundações, cada uma de acordo com a sequência de cartas do mesmo naipe desde o ás (valor = 1) até ao rei (valor = 13). A Figura 1 mostra a configuração inicial do tabuleiro.

Figura 1: Solitário

Considere a seguinte formalização do jogo Solitário em Alloy:

```
enum Vermelhas { Ouros, Copas }
enum Pretas { Paus, Espadas }
enum Estado { Cima, Baixo }

sig Carta {
  valor: Int,
  naipe: Vermelhas + Pretas,
  estado: Estado
}

abstract sig Sequencia { cartas: Int -> Carta }
abstract sig Fila, Fundacao, Pilha, Lixo extends Sequencia {}
one sig F1, F2, F3, F4, F5, F6, F7 extends Fila { }
one sig M1, M2, M3, M4 extends Fundacao { }
```

a. Formalize o facto “todas as cartas de uma fundação têm o mesmo Naipe”.

```
fact allFoundationsHaveSaveNaipe {
   all m: Fundacao | all disj c1, c2 : m.cartas[Int] | c1.naipe = c2.naipe
}
//OR
fact allFoundationsHaveSaveNaipe {
   all m: Fundacao | one m.cartas[Int].naipe
}
```

b. Escreva um predicado de fim de jogo (i.e., todas as fundações estão preenchidas).

```
pred EndGame {
   all m: Fundacao | #(m.cartas[Int]) = 13
}
```

c. Assuma que existe uma função “fun allCardsTurnedUp[f: Fila]: set Carta” que recebe uma Fila como parâmetro e retorna as cartas dessa fila que estão viradas para cima. Formalize o predicado “Inicio de jogo”, em que todas as filas têm apenas uma carta virada para cima.

```
pred InitGame {
   all f: Fila | one (f.cartas :> allCardsTurnedUp[f])
}

pred InitGame {
   {all f: Fila | one allCardsTurnedUp[f]}
}
```

d. Escreva a função (fun allCardsTurnedUp[f:Fila]: set Carta) que, recebendo uma Fila como parâmetro, retorne as cartas que estão viradas para cima.

```
fun allCardsTurnedUp[f: Fila]: set Carta {
  ((f.cartas).(estado.Cima)).(f.cartas)
}
//OR
fun allCardsTurnedUp[f: Fila]: set Carta {
  f.cartas[Int] <: (estado :> Cima).Estado
}
fun allCardsTurnedUp[f: Fila]: set Carta {
  {c:f.cartas[Int]|c.estado=Cima}
}
```

9. Para cada uma das perguntas abaixo, assinale com uma cruz a resposta verdadeira. Cada resposta correcta vale 1 valor. Cada resposta errada desconta 0.5 valores.

9.a). Considere a restrição "**`no x: univ	x->x in r`**". Qual das seguintes expressões é equivalente?	
a. `no iden & r`	x
b. `univ in r.univ`	
c. `r.r in r`	
d. Todas as anteriores	
e. Nenhuma das anteriores	

9.b) Considere a especificação "**`sig X { r: lone X } fact { no x: X	x in x.^r }`**". Qual das seguintes relações satisfaz o modelo?	
a. `X = {(X0, X2), (X1, X2)}`	x
b. `X = {(X1, X2), (X2, X2)}`	
c. `X = {(X3, X2), (X2, X1), (X1, X3)}`	
d. Todas as anteriores	
e. Nenhuma das anteriores	

10. Chama-se de quadrado mágico a uma matriz de números diferentes cuja soma de todas as células de qualquer linha ou coluna resulta sempre no mesmo valor. Por exemplo, no quadrado mágico da figura ao lado, a soma de todas as células, numa linha ou coluna, é sempre 15. Neste problema decidiu-se usar o Alloy Analyzer para encontrar soluções possíveis para quadrados mágicos de dimensão 3.

4	3	8
9	5	1
2	7	6

a. Suponha que cada célula possui duas adjacências (direita e baixo) e um valor numérico. Formalize a assinatura *Cell*.

```
sig Cell { right, below: lone Cell, val: Int }
```

b. Formalize o facto: "Todas as células possuem um valor numérico diferente".

```
fact { no disj s, s': Cell | s.val = s'.val }
```

c. Suponha que as adjacências foram preenchidas manualmente. Formalize a asserção: “Nenhuma relação de adjacência é transitiva”. Escreva um teste que verifique essa asserção.

```
//A solução mais compacta seria a seguinte, que também garante anti-reflexividade e
//anti-simetria:

check { no s: Cell | s in s.^(right+bellow) } expect 0

//---------------------------------------
//A solução mais literal seria a seguinte:

check { no disj c1, c2, c3: Cell | c2 in c1.(right + below) and c3 in c2.(right +
below) and c1 in c3.(right+below) } expect 0
```

d. Escreva uma função *BottomRightCell* que devolve o valor da célula mais abaixo à direita.

```
fun BottomRightCell[]: int { (Cell - (right + bellow).Cell).val }
//or
fun BottomRightCell[]: int {c:Cell | no c.(right+below).square}.val}
```

e. Escreva uma função *TopLeftCell* que devolva o valor da célula mais acima à esquerda. Se não respondeu à pergunta anterior, pode considerar que *BottomRightCell* se encontra corretamente implementada.

```
fun TopLeftCell[]: int { (Cell - Cell.(right + bellow)).val }
```