Mini-test. Consultation restricted to the Alloy slides, manuals and tool. Duration: 100 minutes. 29/nov/2016

Part I

Mark the correct answer with a cross (X). Correct answers earn 1 point. Incorrect ones lose 0.33 points.

1. There are five oceans on Earth, named Artic, Atlantic, Indian, Pacific and Southern. Which of the following **is not** a correct encoding in Alloy?

- ☐ `enum Ocean {Artic, Atlantic, Indian, Pacific, Southern}`
- ☐ `abstract sig Ocean{}`
  `one sig Artic, Atlantic, Indian, Pacific, Southern extends Ocean{}`
- ☐ `one sig Ocean{}`
  `sig Artic, Atlantic, Indian, Pacific, Southern extends Ocean{}`
- ☐ `sig Ocean{}{#Ocean = 5}`[1]
  `one sig Artic, Atlantic, Indian, Pacific, Southern extends Ocean{}`

2. A country may have other countries (but not itself) as neighbors. Which of the following **is not** a correct encoding in Alloy?

- ☐ `sig Country{neighbors: set Country - this}`
- ☐ `sig Country{neighbors: set Country} {this not in neighbors}`
- ☐ `sig Country{neighbors: set Country}  fact {no neighbors & iden}`
- ☐ `sig Country{neighbors: disj set Country}`

3. If a country x is a neighbor of country y, then y is also a neighbor of x. Which of the following **is not** a correct encoding in Alloy?

- ☐ `fact {neighbors = ~neighbors}`
- ☐ `fact {all x, y: Country | x in y.neighbors => y in x.neighbors}`
- ☐ `fact {all x: Country | no y: Country | x->y in neighbors and y->x in neighbors}`
- ☐ `fact {no x, y: Country | x in neighbors[y] and y not in neighbors[x]}`

4. Assuming the definitions `sig Color{}` and `sig Country{neighbors: set Country, color: Color}`, which of the following facts **does not** ensure that neighbor countries have different colors?

- ☐ `fact {all x, y: Country | x->y in neighbors => x.color != y.color}`
- ☐ `fact {all x: Country | x.color not in x.neighbors.color}`
- ☐ `fact {~color.neighbors.color not in iden}`
- ☐ `fact {no color.~color & neighbors}`

5. Assume that all 44 European countries and their neighbors have been described (`one sig Portugal extends Country{}{neighbors = Spain}`, etc.), and one wants to check if it is possible to color them with 4 colors, such that neighbor countries get different colors. What of the following is not a valid command?

- ☐ `run {} for 44 but 4 Color`
- ☐ `assert fourColors{#Color = 4}  check fourColors for 44`
- ☐ `check {} for 44 but 4 Color`
- ☐ `fact fourColors {#Color = 4}  run {} for 44`

[1] To check in Alloy, you need to use 4 or more bits, as in *run{} for 4 Int.*

Mini-test. Consultation restricted to the Alloy slides, manuals and tool. Duration: 100 minutes. 29/nov/2016

6. Which of the following functions <u>does not</u> correctly retrieve the countries with a given color?

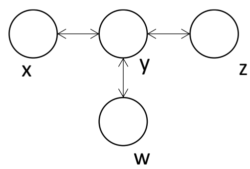
- ☐ `fun countries[c: Color]: set Country {color.c}`
- ☐ `fun countries[c: Color]: set Country {{x: Country | c = x.color}}`
- ☐ `fun countries[c: Color]: set Country { color :> c }`
- ☐ `fun countries[c: Color]: set Country {{x: Country | x->c in color}}`

7. Given relations `R={(a,b), (b,a), (b,c)}` and `S={(a,a),(c,d)}` and set `T ={a}`, which of the following <u>is not</u> true?

- ☐ `(R ++ S) :> T = {(a,a)}`
- ☐ `^R = {(a,b), (a,c), (b,a), (b,c)}`
- ☐ `R.S = {(b,a),(b,d)}`
- ☐ `R.T = {b}`

Part II

Fill in the blanks. In case of multiple choice questions, incorrect answers discount 1/3 of the points.



1. Let's address the problem of moving objects in a network, such as trains in the blocks of a railway network. We start by modeling networks with Alloy. A *network* is here defined as a collection of places and connections between places. On the right, it is shown an example of a network with places labeled *x*, *y*, *z* and *w*, and several bidirectional connections. Complete the Alloy model below.

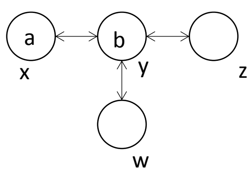
```
sig Place { }

sig Network {
    places: set Place,
    connections: places -> places
}{
    -- Connections are bidirectional: if there is a connection from X to Y, then there is
    -- also a connection from Y to X, i.e., the 'connections' relation is symmetric.
a)1.0    connections = ~connections

    -- A place cannot be connected to itself, i.e., the 'connections' relation is anti-reflexive.
b)1.0    no connections & iden

    -- The network must be connected, that is, there must exist a path between
    -- any two places in the network.
c)1.0    all p1, p2: places | p2 in p1.*connections
}
```

<u>Hint</u>: You can generate examples with the *run{}* command to check your answers. For a better visualization, you can project over Network.



2. Objects may be placed in the places of a network, as illustrated on the right for objects *a* and *b*. A *placement* is here defined as the positioning of a set of objects in a network, such that there is at most one object per place. Complete the Alloy model below.

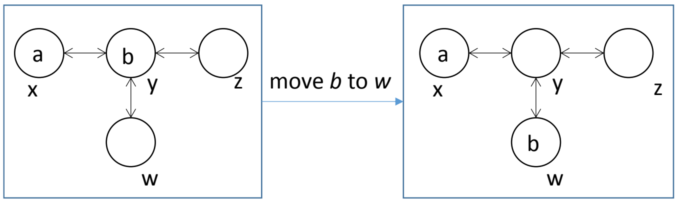
Mini-test. Consultation restricted to the Alloy slides, manuals and tool. Duration: 100 minutes. 29/nov/2016

```
sig Object{}

sig Placement {
      network : Network,
a)1.0 objects: set Object,
      -- positions relates objects with places, such that each object has exactly
      -- one place and each place has at most one object
      positions: objects lone -> one Place
}{                      b)0.5     c)0.5
      -- The places where objects are positioned must belong to the network.
d)1.0 positions[objects] in network.places
}
```

Hint: You can generate examples with the *run* command to check your answers. For a better visualization, you can project over Network and Placement.

3. Objects may be moved one at a time to immediately adjacent places. In the example on the right, an object is moved to an adjacent position, originating a new placement. The only other legal movement would be to move object *b* to place *z*. Complete the following Alloy specification of the *moveObject* operation.



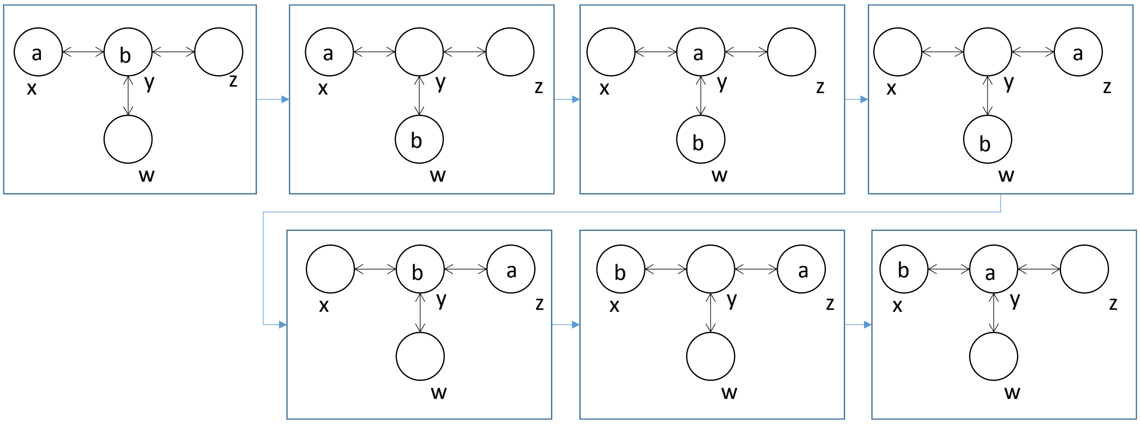

```
-- Moves an object o to an adjacent place p in a placement t,
-- resulting in a new placement t'.
pred moveObject[t: Placement, o: Object,  p: Place, t': Placement] {
      -- Pre-conditions:
      -- the object (o) must exist in the initial placement (t)
a)1.0 o in t.objects
      -- the target place (p) must be unnocupied in the initial placement (t)
b)1.0 no t.positions.p
      -- the target place (p) must be adjacent to the initial place of the object (o)
c)1.0 t.positions[o] -> p in t.network.connections

      -- post-conditions (one per field of t')
      t'.network = t.network
d)2.0 t'.objects = t.objects
      t'.positions = t.positions ++ o -> p
}
```

Hint: You can generate examples with the *run moveObject* command to check your answers.

4. Next we use Alloy to generate sequences of placements of objects in a network, by moving one object in each step. The following example shows a minimal sequence of movements/placements to swap two objects. In this example, 6 movements (and a total of 7 placements) are needed.

Mini-test. Consultation restricted to the Alloy slides, manuals and tool. Duration: 100 minutes. 29/nov/2016



We start by imposing an ordering of the generated instances of Placement, with the following directive in the begin of the Alloy model.

```
open util/ordering[Placement]
```

Next you have to write a fact to impose that subsequent instances of Placement correspond to valid applications of the *moveObject* operation. Note that for any instance *t* of Placement, the next instance in the sequence is denoted *t.next* (the first and last instances are named *first* and *last* respectively).

a)2.0

```
fact {
    all t: Placement, t' : t.next | some o: Object, p: Place | moveObject[t, o, p, t']
}
```

Hint: To check your model, you can use the following Alloy code, corresponding to the above example.

```
one sig x, y, z, w extends Place {}
one sig n extends Network {} {
    places = x + y + z + w
    connections = x->y + y->x + y->z + z->y + y->w + w->y
}
one sig a, b extends Object {}
one sig initial extends Placement {} {
    network = n
    objects = a + b
    positions =  a->x + b->y
}
-- Swap objects in a minimal number of steps (6 moves, 7 Placements)
run success {
   first = initial and last.positions = a->y + b->x
} for 7 but exactly 1 Network, 2 Object, 4 Place

-- Trying to swap objects in fewer steps should fail
run failure {
   first = initial and last.positions = a->y + b->x
} for 6 but exactly 1 Network, 2 Object, 4 Place
```