This is a specimen mini-test.

Note: Structure of the mini-test will be identical, but not necessarily the same. Group I might be replaced by true/false questions, instead.

Duration: 90 minutes **20.0**

Student Number:

Name:

This test is composed of 12 pages. Identify yourself on all the pages. Each answer should be given using solely the blank space that follows it.

GOOD LUCK!

Group I (3.0)

1. Consider the following Alloy models:

(A)
```
abstract sig A, B {}
one sig A1, A2, A3 extends A {}
one sig B1, B2 extends B {}
one sig R { r : A some->lone B }{ r[A1] = r[A2] && r[A1] = B1}
```

(B)
```
abstract sig A, B {}
one sig A1, A2, A3 extends A {}
one sig B1, B2 extends B {}
one sig R { r : A lone->lone B }{ r.B1 = r.B2 }
```

(C)
```
abstract sig A, B {}
one sig A1, A2, A3 extends A {}
one sig B1, B2 extends B {}
one sig R { r : A some->some B }{ r.B1 = r.B2 }
```

(D)
```
abstract sig A, B {}
one sig A1, A2, A3 extends A {}
one sig B1, B2 extends B {}
one sig R { r : A lone->some B }{}
```

(E)
```
abstract sig A, B {}
one sig A1, A2, A3 extends A {}
one sig B1, B2 extends B {}
one sig R { r : A lone-> B }{ A1] = r[A2] && r[A1] = B1 }
```

(F)
```
abstract sig A, B {}
one sig A1, A2, A3 extends A {}
one sig B1, B2 extends B {}
one sig R { r : A some->lone B }{ r.B1 = r.B2 }
```

For each of the models above, say whether the command
`run {} for 4`
finds an instance. In case an instance is not found, explain why.

(continuation of question II.1)

Group II (1.0+3.0)

1. Consider the following Alloy model:

```
sig Node {}
sig DAG {
  edge: Node->Node
}

pred add (g,g': DAG, e: Node->Node) {
  one e
  g'.edge = g.edge + e
}

fact acyclic {all g: DAG | no n: Node | n in n.^(g.edge)}

run add for 3 but 2 DAG
```

   (a) Which are the analysis variables for the command `run add for 3 but 2 DAG`?

(continuation of question II.1)

(b) Which is the analysis constraint for the command
`run add for 3 but 2 DAG`?

Group III (3.5+5.0)

1. Consider the following Alloy model:

```
sig Tree { root : lone Node }
sig Node {
  left, right : lone Node,
  value : one Int,
}
```

The signature `Tree` represents a set of binary trees. Each tree has a root, which is zero or one nodes. Each node has an integer value and a left and a right sub-tree.

(a) Define a function `parent` that defines the parent relation: Node p is parent of n iff n is a (left or right) child of p.

Define a predicate `wellFormedTree` that says that

(b) for all nodes, if at least one of the sub-trees is non-empty then the two sub-trees are not equal,

(c) the parent relation has no cycles,

(d) every node, except the root node, has one parent,

(e) for all nodes, the values in the left sub-tree are less than the value in the node and the value in the node is less than the values in the right sub-tree,

(f) no two different nodes have the same value,

(g) for all nodes the difference between the number of nodes in the left and right sub-tree is less than or equal to 1.

(continuation of question III.1)

2. Consider the following Alloy model:

```
one sig List {
  head : lone Node
}

sig Node {
  next : lone Node,
  value : Int
}
```

The signature `List` represents a list. The list has a head, which is zero or one nodes. Each node has an integer value and a field `next`, which is zero or one nodes.

Extend/change the model as follows:

(a) Add a notion of time and make sure that there is an ordering on that time.

(b) Change the signatures `List` and `Node` so that `head` and `next` are time-dependent.

(c) Restrict the model so that the list initially is empty.

(d) Define an operation `delete` that, given a node `n`, removes `n` from the list. It should not be possible to remove a node that is not in the list. Make sure that this operation does not change anything else in the list: The nodes before and after `n` must be kept in the list and their mutual order must not be altered.

(e) Restrict the model with a notion of traces. This restriction should say that at each time, except for the last time, it is possible to either insert or delete a node. (You can assume that the operation `insert` is already defined and that `insert` inserts in order.)

(f) Consider the invariant *the list is always sorted with respect to the integer values*. Define this invariant and write a command that checks whether the invariant holds (within a certain scope).

(continuation of question IV.2)

Group IV (1.5 + 1.5 +1.5)

1. What is skolemization? When does Alloy use skolemization and which are the advantages?
2. Is the Alloy logic decidable? What does it mean for a logic to be decidable?
3. Explain how the Alloy analyzer uses the notion of scope.

(continuation of question IV.)

END