Total: 99/100

David Pick
CM 2403
Formal Methods

OK 2a) Holds

OK 2b) p = {Thing$1->Thing$0}, q = {Thing$0->Thing$0}, s = {Thing$0, Thing$1}
(s.(p - q) = {Thing$0, Thing$1}) != (s.p - s.q = {})

OK 2c) p = {Thing$1->Thing$0, Thing$1->Thing$1}, q = {Thing$0->Thing$0, Thing$0->Thing$1},
s = {Thing$0, Thing$1}
(s.(p & q) = {}) != (s.p & s.q = {Thing$0, Thing$1})

(-1) Model missing from submission doc but present in repository

3)

OK

```
// Problem A.1.5
module homework/hw3_2

sig Node {}

//each node has 1 parent except for root node
//root node has no parent
//there is only 1 root node
//two nodes can't have the same children
//you can reach all nodes from the root node
//nodes can have 0 or more children
pred isTree[r: Node -> Node] {
//one n1: Node |  n1->n1 in r //1 node without parent
//two nodes can't have the same child
all n1, n2: Node | n1->n2 in r implies not n2->n1 in r

all n1, n2, n3: Node |
n1->n2 in ^r and n2->n3 in r implies not n3->n1 in r

one n1: Node | no n2: Node | n2->n1 in r

all n1: Node | lone n2: Node | n2->n1 in r
}

run isTree for 4
```

4)

```
// Problem A.1.10
// Written by David Pick
module homework/hw3_3

abstract sig Name {
  address: set Addr + Name
}

sig Alias, Group extends Name {}
sig Addr {}
```

```
    // invariants
    fact {
      // Put your answers to a and b here, plus the
      // additional invariant noted below.

       //there is no name, that has itself in the address
OK  all n: Name | not n in n.^address

    //all names eventually map to an address
    //All names (in the domain of the address relation) eventually map to an address.
    all n: address.univ |
OK  one a: Addr |
    a in n.^address

        //alias's must map to only 1 address or name
OK  all a: Alias | #a.^address < 2
    }

    // simulation constraints
    pred show[] {
      // Put your answers to c and d here.

      // You may want to add additional constraints to make
      // the generated instances more interesting.

      //the address book has at least two levels.
      //a name must be connected to a name, which is then connected to
      //either another name or and address
OK    some n1, n2: Name, a: Addr | n1->n2 in address and n2->a in address

      //some groups are non-empty
OK    some g: Group | some g.address
      some g: Group | no g.address
      //Group.address :> Name
      //
      //Alias.^address
    }

    run show for 3 but 6 Name
```

OK 4 e) Group.address :> Name

OK 4 f) Group-address.univ

OK 4 g) Alias.^address

6) 4 hours