Exercises

3.1 The following Alloy model constrains a binary relation to have a collection of standard properties:

```
module properties

pred show () {
   some r: univ -> univ {
--     some r            -- non empty
       r.r in r          -- transitive
       no iden & r       -- irreflexive
       ~r in r           -- symmetric
       r.~r in iden      -- functional
       ~r.r in iden      -- injective
       univ in r.univ    -- total
       univ in univ.r    -- onto
       }
   }
run show for 4
```

A finite binary relation cannot have all these properties at once. Which individual properties, if eliminated, allow the remaining properties to be satisfied? For each such property eliminated, give an example of a relation that satisfies the rest.

You can use the Alloy Analyzer to help you. The run command instructs the analyzer to search for an instance satisfying the constraints in a universe of at most 4 atoms. To eliminate a property, just comment it out (as done for *non empty* above).

If you're not yet comfortable with the relational calculus style, you can try a reformulate the properties with quantifiers. For example, you might prefer this definition of totality:

```
all x: univ | some x.r
```

which you could check with an assertion like this:

```
assert Same {
   all r: univ -> univ |
      univ in r.univ iff (all x: univ | some x.r)
   }
check Same
```

3.2 When writing 'navigation expressions', it's tempting to simplify them, but not all simplifications are valid. For a given set s and binary relations p and q, which of the following algebraic properties hold? If the property doesn't hold, show a counterexample.

a. distributivity of join over union: s.(p+q) = s.p + s.q

b. distributivity of join over difference: s.(p-q) = s.p - s.q

c. distributivity of join over intersection: s.(p&q) = s.p & s.q

Here's an example of how you might check the first using the Analyzer:

```
module distribution
assert union {
   all s: set univ, p, q: univ -> univ | s.(p+q) = s.p + s.q
 }
check union for 4
```

The command tells the analyzer to find a counterexample within a universe of 4 elements. When you find that a property does not hold, try and obtain the smallest counterexample you can, by reducing the scope (eg, replacing for 4 by for 2), or by adding additional constraints (eg, #p < 2).

3.3 A *tree* is a relation that satisfies some properties. What exactly are the properties? Express them in relational logic, and illustrate with a few examples.

Here is a template to help you:

```
module tree
pred isTree (r: univ -> univ) { … }
run isTree for 4
```

You can replace the ellipsis by some constraints on the relation r, and execute the command to visualize some sample instances. You may need to add some constraints to make the instances nontrivial.

3.4 A *spanning tree* of a graph is a subgraph that's a tree and covers all its nodes. Make this definition precise, and give an example of a graph with two distinct spanning trees. Here is a template to help you:

```
module spanning
pred isTree (r: univ -> univ) { … }
pred spans (r1, r2: univ -> univ) { … }
pred show (r, t1, t2: univ -> univ) {
   spans (t1, r) and isTree (t1)
   spans (t2, r) and isTree (t2)
   t1 not = t2
   }
run show for 3
```

Spanning trees have many uses. In networks, they're often used to set up connections. In the Firewire protocol, for example, a spanning tree is automatically discovered, and the root of the tree becomes a leader that coordinates communication.

3.5 Suppose you are modelling each of the following relationships as a binary relation. Say for each what properties (functional, injective, reflexive, symmetric, transitive) you would expect the relation to have, clarifying the relation's meaning when necessary:

a. the sibling relationship, between children with the same parents;

b. the links relationship, between a host on a network and the hosts it is linked to;

c. the contains relationship, between a directory in a file system and its contents;

d. the group relationship, between graphical elements in a drawing program and groups (collections of elements that are selected and deselected together);

e. the sameGroup relationship, between graphical elements in the same group;

f. the supercedes relationship, between a file in one file system and a file in another file system, which holds when the first file is a newer version of the second file.

3.6 Suppose we model the map of the London Underground as follows. A railway line R is modelled as a separate relation R_o for each direction it goes in, with a suffix o indicating the compass orientation, and the stations are represented by a set Station with

a subset `L` corresponding to the stations of each line `L`, and a scalar for each station bearing its name.

For example, the Central Line runs east-west, so it will be represented by two relations `Central_East` and `Central_West`, and a set of stations `Central`, which will include the station `BondStreet`. Separate branches of a line are not modelled by separate relations, so if a line branches, then in the direction of the split, there will be a station mapped to more than one station.

Formalize each of these constraints, using comprehensions and quantification as little as you can:

a. If you go westbound or eastbound on the Circle Line, you will eventually get back to where you started.

b. You can change from the Jubilee Line to the Central Line at exactly one station.

c. Aldgate is the last station on the eastbound Metropolitan Line.

d. The Metropolitan Line westbound splits into two branches at Harrow on the Hill.

e. The District Line westbound has two more splitting points than the southbound East London Line.

f. The Central Line eastbound splits at some point into two branches that come together again.

g. The Jubilee Line never splits, and the sequence of stations in one direction is the exact reverse of the sequence of stations in the other direction.

h. To get from St. John's Wood to Oxford Circus, you can take the Jubilee Line southbound (to Bond Street) and then take the Central Line eastbound, or you can take the Jubilee Line southbound (to Baker Street) and then take the Bakerloo Line southbound.

Incidentally, you can find a map of the Underground online at *http://tube.tfl.gov.uk/*. You don't need it to complete this problem. You also don't need to use the Alloy Analyzer, although you might find it helpful to simulate some constraints.

3.7 Transitive closure is not axiomatizable in first-order logic. In short, that means that if you want to express it, you need a special operator, because it can't be defined in terms of other operators. Here's a bogus attempt to do just that; your challenge is to use the Alloy Analyzer to find the flaw.

Recall that the transitive closure of a binary relation r is the smallest transitive relation R that includes r. Let's say R is a transitive cover of r if R is transitive and includes r. To ensure that R is the smallest transitive cover, we can say that removing any tuple a->b from R gives a relation that is *not* a transitive cover of r. Formalize this by completing the following template:

```
module closure

pred transCover (R, r: univ -> univ) { … }
pred transClosure (R, r: univ -> univ) {
   transCover (R, r) and …
   }

assert Equivalence {
   all R, r: univ -> univ | transClosure (R, r) iff R = ^r
   }
check Equivalence for 3
```

Now execute the command, examine the counterexample, and explain what the bug is. The official definition of UML 1.0 had this problem [].