A.2 Extending Simple Models

The problems in this section exercise the full Alloy language, but give you the initial structure of the model.

A.2.1 Telephone Switch Connections

Consider the following model of connections in a telephone network:

```
module exercises/phones

sig Phone {
   requests: set Phone,
   connects: lone Phone
   }
```

The signature *Phone* represents a set of telephones. For a phone *p*, *p.requests* is a set of phones that *p* is requesting connections to, some of which may have been granted, and *p.connects* is the phone that *p* is currently connected to (or none).

(a) Simulate the model by adding a predicate and running it. Add some constraints to the predicate to ensure that you don't get boring cases; for example, you might say that there should be some requests and some connections.

(b) Add two invariants: that every connection has a matching request (on the assumption that requests don't disappear until the connections they spawned are torn down), and that there are no conference calls (in which a phone is involved in more than one connection).

(c) Now incorporate call forwarding, by extending the state with a new relation *forward* from phones to phones, where *p.forward*, if nonempty, is the phone that an incoming call to *p* should be forwarded to. Change the constraint relating *requests* and *connects* to account for forwarding. Simulate some interesting examples of call forwarding, adding some extra simulation constraints if necessary.

A.2.2 Invariant Preservation in an Address Book

In this problem, you are given a model of a simple address book program, with operations to add, delete,viol and look up a name, and an invariant characterizing the address book's well-formedness properties.

(c) Generate some examples of executions of the operations, again by adding and running appropriate predicates.

(d) Find counterexamples showing that neither *add* nor *delete* preserves the invariant. To do this, you'll need to define assertions that invoke the operations and the invariant, and commands to check them. Reduce the scope if necessary to obtain the smallest counterexample possible.

(e) Elaborate the two operations with additional preconditions—constraints on the prestates—that ensure the invariant is preserved, and rerun the preservation check to show that you have succeeded. Increase the scope to give you more confidence, and briefly justify your choice of scope.

(f) Rerun your simulations from (c) to check that you haven't inadvertently overconstrained the operations.

A.2.3 Inmate Assignments

A program is needed to assign inmates to cells in a prison. The assignment must avoid placing two inmates in the same cell if they are members of different gangs.

Here is a suitable template:

```
module exercises/prison

sig Gang {members: set Inmate}
sig Inmate {room: Cell}
sig Cell {}

pred safe () {
  ... your constraints here
    }

pred show () {
  ... your constraints here
  }
run show
```

(a) Complete the predicate *safe* characterizing a safe assignment, and generate examples of both safe and unsafe assignments by running the simulation predicate *show*, with appropriate invocations of *safe* as its constraint.

(b) Write a new predicate called *happy*, saying that gang members only share cells with members of the same gang. A safe assignment is not necessarily a happy assignment. By writing an assertion and a command to check it, find a counterexample, and explain why not.

(c) Add a constraint as a fact that ensures that safety will indeed imply happiness. Run your simulation predicate again to make sure that you haven't introduced an inconsistency, and check the assertion again to make sure it now has no counterexample.

A.3 Classic Puzzles

The exercises in this section give practice in writing Alloy and structuring small models.

♥*A.3.1 A Surprising Syllogism*

A song by Doris Day goes

> "Everybody loves my baby
> but my baby don't love nobody but me."

David Gries has pointed out that, from a strictly logical point of view, this implies "I am my baby." Check this, by formalizing the song as some constraints, and Gries's inference as an assertion. Then modify the constraints to express what Doris Day probably meant, and show that the assertion now has a counterexample.

♥*A.3.2 Ceilings and Floors*

A song by Paul Simon goes

> "One man's ceiling is another man's floor."

Does this imply that one man's floor is another man's ceiling? Formalize the two constraints in Alloy, and check an assertion that the first implies the second. If you get counterexamples that don't make sense because of implicit assumptions, add them as new constraints, and check again.

A.3.3 Barber Paradox

Consider the set of all sets that do not contain themselves as members. Does it contain itself? This paradox was discovered by Bertrand Russell in 1901, and revealed an inconsistency in Frege's naive set theory.