Problem Set 1: Small Alloy Exercises

1 Extending Simple Models

1.1 [Easy] 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 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. Does this condition always follow from a safe assignment? By writing an assertion and a command to check it, find a counterexample.

c. Add a constraint as a fact that ensures that safety will indeed imply happiness. Run the simulation predicate to make sure

that you haven't introduced an inconsistency, and check the assertion again to make sure it now has no counterexample.

2 Classic Puzzles

2.1 [Easy] 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.

2.2 [Easy] **Russell's Paradox**. A popular form of Russell's paradox asks: in a village in which the barber shaves every man who doesn't shave himself, who shaves the barber?

Here's a statement of the paradox in Alloy:

```
module russell
sig Man {shaves: set Man}
one sig Barber extends Man {}
fact {
   Barber.shaves = {m: Man | m not in m.shaves}
   }
```

Explore this paradox in Alloy, as follows:

a. Use the analyzer to show that this model is inconsistent, at least for a village of small size.

b. Feminists have noted that the paradox disappears if the existence of women is acknowledged. Make a new version of the model that classifies villagers into men (who need to be shaved) and women (who don't), and show that there is now a simple solution.

c. A more drastic solution, noted by Edsger Dijkstra [1], is to allow the possibility of there being no barber. Modify the original model accordingly, and show that there is now a solution.

d. Finally, try a variant of the original model that allows multiple barbers, who shave any man who doesn't shave himself, and show there is again a solution.

2.3 [Easy] **Halmos's Handshaking**. Here is the famous handshake problem due to the mathematician Paul Halmos, in his own telling:

> *My wife and I were invited to a party recently, a party attended by four other couples. Some of the 10 knew some of the others and some did not, and some were polite and some were not. As a result, a certain amount of handshaking took place in an unpredictable way, subject only to two obvious conditions: No one shook his or her own hand and no husband shook his wife's hand. When it was all over, I became curious, and I went around the party asking each person: "How many hands did you shake? And you? And you?" What answers could I have received? Conceivably, some people could have said None, and others could have given me any number between 1 and 8 inclusive. That's right isn't it? Since self-handshakes and spouse-handshakes were ruled out, eight is the maximum number of hands that any one of the party of 10 could have shaken.*
>
> *I asked nine people (everybody, including my own wife), and each answer could have been any one of the nine numbers 0 to 8 inclusive. I was interested to note, and I hereby report, that the nine different people gave me nine different answers: someone said None, someone said One, and so on and, finally, someone said Eight. Next morning, I told the story to my colleagues, and I challenged them, on the basis of the information just given, to tell me how many hands my wife shook.*

a. Solve the problem by modelling it in Alloy, and using the Analyzer to find a solution. Solving for 10 people will take longer than solving for 4 or 6, so use a smaller number until your confident that your model makes sense. (If you don't want the fun of solving it yourself, you can use the solution in the standard Alloy distribution).

b. Might there be another solution, in which Halmos's wife shook a different number of hands? Extend your model to allow this to be checked. You might want to refactor it a bit so that the two candidate solutions don't lead to two sets of almost identical constraints.

3 Metamodels

The exercises in this section give practice in constructing metamodels, which some people find confusing. A metamodel is a model like any other, and need not show the qualities of the models it captures. For example, a metamodel of state machines doesn't have to be dynamic itself: a state machine is just a structure that can be imbued with a dynamic interpretation.

3.1 [Easy] **State Machine Definition**. A *state machine* has one or more initial states, and a transition relation connecting each state to its successors. Construct an Alloy model of a state machine, and, by adding constraints and having the analyzer solve them, generate a variety of examples of machines:

a. A *deterministic* machine, in which each state has at most one successor;

b. A *non-deterministic* machine, in which some states have more than one successor;

c. A machine with *unreachable* states;

d. A machine without *unreachable* states;

e. A *connected* machine in which every state is reachable from every other state;

f. A machine with a *deadlock*: a reachable state that has no successors;

g. A machine with a *livelock*: the possibility of an infinite execution in which a state that is always reachable is never reached.

4 Small Models

The exercises in this section involve the construction of small models in well-defined settings.

4.1 [Hard] **Unix file system**. In the Unix file system, each file is represented by an *inode*. The inode includes some basic properties of the file (permission bits, file type, etc.), and has a sequence of 10 addresses that point to disk blocks containing the file's data.

In addition, there are three further *indirect* addresses. The first involves one extra level of indirection: it points to a block containing addresses, rather than data, of blocks which hold the data. The second involves two levels of indirection: it points to an address block that point to address blocks that point to data blocks. The third involves three levels.

All the inodes are stored in an array called the *inode table*. The index of a given inode in this array is its *inumber*. A directory is represented as a file whose data consists of a list of inumber/filename pairs. The root directory is associated with some fixed inumber.

To locate a file, you start at the root directory, and look up the prefix of the file's pathname. This gives an inumber, which you look up in the inode table. The inode obtained is either the file required (if no more of the pathname remains), or another directory, for which the process is repeated (on the rest of the pathname).

a. Start by building a model of inodes, inumbers and blocks. Ignore indirect addressing. Explore some sample structures by writing simulation constraints, adding any invariants that you discover you omitted.

b. Build a model of pathnames, treating a pathname as list, consisting of a name (the first element) and a pathname (the rest). Explore some sample pathnames by writing simulation constraints, adding any invariants that you discover you omitted.

c. Now you're going combine the two parts of your model, and define a function that models lookup: given a pathname, it returns a set of inodes. You'll want to define lookup recursively, but Alloy functions cannot be recursive. Instead, you

can declare a relation corresponding to the lookup, which is used without recursion in the function, but is itself defined by a recursive constraint.

d. Formulate and check two assertions: that each pathname resolves to at most one inode, and that no two distinct pathnames resolve to the same inode. Which of these did you expect to hold? If your analysis reveals flaws in your model, correct them.

e. Finally, add the notion of indirect addressing. Try to do it in a modular fashion, with as little disruption as possible to your model of name lookup.

References

[1] Edsger W. Dijkstra, *Where is Russell's Paradox?*, EWD-923A, 22 May, 1985; available online at http://www.cs.utexas.edu/users/EWD.