CITS5501 Software Testing and Quality Assurance
Semester 1, 2020
Workshop 8 – Formal specifications

1 Requirements

Identify some problems with the following requirements:

a. "The system shall not accept passwords longer than 15 characters."
b. "The software shall support a water level sensor."

2 Alloy sigs and properties

How would you translate the following into Alloy syntax? All of these can be done by declaring *sigs* and *properties*.

a. There exist such things as chessboards.
b. There is one, and only one, tortoise in the world.
c. There exists at least one policeman.
d. Files have exactly one parent directory.
e. Directories have at most one parent directory.
f. Configuration files have at least one section.

3 Alloy facts

Recall that in Alloy, *facts* are additional constraints about the world, that aren't expressed in the sigs, and can be used to "tighten" the meaning of your model. (Some constraints could be expressed either in the sig, or as a fact.) For instance, using the example `File` and `Dir` and `FSObject` sigs from the lecture:

```
fact {
  File + Dir = FSObject
}
```

means, "the set of files, plus the set of directories, is the same as the set of all file-system objects".

Or, if we give our fact a name:

```
fact noOtherFSObjects {
  File + Dir = FSObject
}
```

Take a look at the Alloy quick reference, which gives other operators you can use besides "`+`" and "`=`". For instance, "`-`" (set subtraction), "`#`" (set cardinality, or "size"), "`&`" (set intersection), "`in`" (set membership), typical comparison operators ("`<`", "`>`", "`=<`", "`=>`"), and typical logical operators ("`&&`", "`||`", "`!`"), and see if you can give Alloy facts which express the following.

a. Assume we have a sig `LectureTheatre{}` and a sig `Venue{}`.
   Give a fact which constrains every lecture theatre to also be a venue. (Note that we could do this using "extends" in the sig, also. But sometimes it's more convenient to express things using facts, or the constraint we want is too complicated for just "extends".)
b. Assume we have the sigs `Carnivore{}`, `Omnivore{}`, `Herbivore{}`. Write a fact constraining `Omnivore` to be the intersection of `Carnivore` and `Herbivore`.

Sample solutions:

```
// a.

// All lecture theaters are venues
fact { LectureTheatre in Venue }

// However note that you will get a warning in Alloy if you
// try this: by default, each sig is a distinct type.
// For a fully working example:

sig Venue {}
sig Lecture in Venue {}

pred show() {}

run show for 3
```

```
// b.

// Assume 'Herbivore' and 'Carnivore' mean 'does eat plants'
// and 'does eat meat' respectively.

// Omivore is the intersection of Herbivore and Carnivore:
fact { Omivore = Herbivore & Carnivore }

// Again, to avoid warnings and model this fully,
// we'll need to amend the sigs. Full working example:

sig Animal {}
sig Herbivore in Animal {}
sig Carnivore in Animal {}

fact { Omivore = Herbivore & Carnivore }

pred show() {}

run show for 3
```

4 Facts with quantifiers

The facts in the previous section constrain sets (e.g. the set of lecture theatres, or the set of omnivores).

We can also write constraints that apply to every entity *in* some set.

For example, suppose we have the following sigs:

```
sig Activity {}
sig Person { hobbies: set Activity }
sig ComputerScientist extends Person {}
```

We can apply the following constraint: "Computer scientists have no hobbies:"

```
fact {
  all cs : ComputerScientist | #cs.hobbies = 0
}
```

In other words: people can have zero or more hobbies; but for all people who are computer scientists, if we look at their hobbies, the cardinality will be 0.

It’s also possible to write this using “no” (another sort of “multiplicty”, like `lone` or `set`):

```
fact {
  all cs : ComputerScientist | no cs.hobbies
}
```

Try extending this model to say:

a. Economists are also people.
b. Economists have at most one hobby.
c. Students are people.
d. Students have at least one hobby.
e. Bots are not people.

Sample solutions:

a. Economists are also people:

```
// if we assume Economist never overlaps with ComputerScientist
sig Economist extends Person {}
```

b. Economists have at most one hobby.

```
fact {
  all econ : Economist | lone econ.hobbies
}
```

c. Students are people.

```
sig Student extends Person {}
// Alternative to this: plausibly, we might instead model
// students as a subset of Person, so you can
// be a student economist.
```

d. Students have at least one hobby.

```
fact {
  all s : Student | some s.hobbies
}
```

e. Bots are not people.

```
// We can just declare bots as a separate sig --
// by default, they won't overlap with Person
sig Bot {}
```

Further exercises

Model the following systems:

1. An *alarm clock* has two sorts of time it can keep track of: the *current* time, and an *alarm* time.

   It *always* has a current time, and *may* have an alarm time.

2. A person can have up to two parents (who are also people). No person is their own parent. There exists exactly one person – let's call them Bob – who has no parents. (Poor Bob.)

3. Entities called *nodes* exist (we will use them to model linked lists). A node may have a successor node, called its "*next*" node.

4. A *contact list* may contains multiple *entries.* Each entry must have a "personName", and may have one or more telephone, street address, or emails.