Model-checking: basic concepts

First Order Logic

- The logic of Alloy is based upon ***First Order Logic*** (extended with various forms of syntactic sugar).
- The Alloy Analyser uses a technique called ***model-checking*** to analyse specifications written in FOL
- This lecture looks at the basic concepts involved:
  - Structures, signatures, and models
  - Consistency
  - Validity
  - Limited scope model checking
  - The “small scope hypothesis”

First Order Logic

- An Alloy specification consists of a set of logical statements.
- Example (University):

sig Student { }	There are students.
one sig JaneDoe **extends** Student { }	There is a student called Jane Doe.

sig Module {
  class : **set** Student
}
	There are modules.
	There is a relation, class, between Module and Student.

one sig CSC9P6 **extends** Module { }	There is a module called CSC9P6.
fact { JaneDoe in CSC9P6.class }	Jane Doe is in the CSC9P6 class.

Signature of a specification

- Every specification contains some undefined elements (*open variables*). These form the ***signature*** of the specification:
- The signature of the University specification consists of:

Student	We know this is a set of atoms, but not how many there are.
JaneDoe	We know this is a set containing a single atom.
Module	We know this is a set of atoms, but not how many there are.
Class	We know this is a binary relation from Module to Student, but we don't know exactly which tuples are in it.
CSC9P6	We know this is a set containing a single atom.

Structure of a signature

- A ***structure*** of a signature is an assignment of specific values to each of the open variables in the signature.
- For example, here are two possible structures of the signature of the University specification:

```
Student =  {JaneDoe, Student0, Student1}
JaneDoe = {JaneDoe}
Module =   {CSC9P6, Module0}
Class =    {CSC9P6 -> Student0, CSC9P6 -> JaneDoe}
CSC9P6 =  {CSC9P6}
```

```
Student =  {JaneDoe, Student0, Student1}
JaneDoe = {JaneDoe}
Module =   {CSC9P6, Module0}
Class =    {CSC9P6 -> Student0, CSC9P6 -> Student1}
CSC9P6 =  {CSC9P6}
```

Models of a specification

- A ***model*** of a specification is a structure for that specification which makes all the statements in the specification true.
- Which of these two structures is a model of the University specification?

```
Student =   {JaneDoe, Student0, Student1}
JaneDoe = {JaneDoe}
Module =    {CSC9P6, Module0}
Class =     {CSC9P6 -> Student0, CSC9P6 -> JaneDoe}
CSC9P6 =  {CSC9P6}
```

```
Student =   {JaneDoe, Student0, Student1}
JaneDoe = {JaneDoe}
Module =    {CSC9P6, Module0}
Class =     {CSC9P6 -> Student0, CSC9P6 -> Student1}
CSC9P6 =  {CSC9P6}
```

Consistency and Validity

- A logical statement is ***consistent*** if there exists at least one model of that statement. (There is **at least one** possible world in which the statement is true.)
- A logical statement is ***valid*** if every structure is a model of that statement. (The statement is true in **all** possible worlds.)
- Consider each statement below in the context of models of the University specification. Is the statement consistent? Is it valid?
  - **some** s:Student | s **not in** CSC9P6.class
  - **no** s:Student | s **not in** CSC9P6.class
  - **no** s:Student | s **in** CSC9P6.class
  - **some** s:Student | s **in** CSC9P6.class

Model-checking in Alloy

- The Alloy Analyser is able to generate possible models of a specification and examine whether logical statements are true in these models.
- There are two main model-checking commands:
  - **Run** a predicate. This checks whether the predicate is consistent. The analyser tries to find a model in which the predicate is true.
  - **Check** an assertion. This checks whether the assertion is valid. The analyser tries to find a model in which the assertion is false (a counter-example).
- Note the difference! In general,
  - run a predicate to find out if something **can possibly be true**
  - check an assertion to find out if something **must always be true**

Scope

- It is not computationally possible to check every possible model of a specification. (There could be infinitely many models!)
- Therefore, Alloy only checks models up to a certain size (***scope***).
- By default the scope is 3, meaning that Alloy will only consider models containing up to 3 atoms from each set.
- The scope can be made larger, or customized by modifying the run or check command. For example:

```
run show
run show for 5
run show for 5 but exactly 1 Module
```

Limitations of model-checking

- Because of the use of a finite scope, model-checking in Alloy has certain limitations:
- When a predicate is run, if a model instance is found, we can be certain that the predicate is consistent.
  - But if no instance is found, we cannot conclude that the predicate is inconsistent.
  - It *may* be inconsistent…
  - …or, maybe we could find a model instance by using a larger scope.
- Similarly, when an assertion is checked, if a counter-example is found we can be certain that the assertion is invalid.
  - But if no counter-example is found, we cannot be sure that the assertion is valid.
  - It *may* be valid…
  - …or, maybe we could find a counter-example by using a larger scope.

The “small scope hypothesis”

- Despite these limitations, the designers of Alloy argue that it is still a useful tool to use in analysing software systems.
- Their argument is based on the **“small scope hypothesis”**:

> “ ‘Most bugs have small counterexamples.’
>
> That is, if an assertion is invalid, it probably has a small counterexample. I call this the ‘small scope hypothesis’, and it has an encouraging implication: if you examine all small cases, you’re likely to find a counterexample. “
>
> [Jackson, 2011]