Lecture aims

1. Systematically construct tests from formal models using the Test Template Framework.
2. Apply adapted Test Template Framework to Alloy models.

Lecture plan

Introduction to model-based testing

1. Why test in high-integrity systems? Catch mistakes. Test in environment. Validation.
2. Testing can be: ad-hoc, exploratory, or systematic. For high-integrity systems, systematic is essential, exploratory is useful, ad-hoc is almost worthless.
3. Model-based testing: use of design models to generate test artifacts (inputs, oracles, stubs, drivers).
4. Example: triangle program. Show Alloy definition (below)

Test template framework

1. Early model-based testing, which has inspired many new tools and frameworks.
2. It considers testing to be more than just a statement about input data — the functional behaviour specification, the test outputs, and test purpose are all important considerations that must be taken into account.
3. Process: (1) IS, OS, VIS; (2) partition; (3) prune; (4) select inputs; and (5) derive outputs.
4. Show IS, VIS, and OS for the Triangle program.
5. Partitioning tactics:

   (a) Cause-effect analysis: four types of triangle (described using input space)

   (b) Partition analysis:

   ```
   PA_ISO.1 == [CE_ISO | x = y and z != x]
   PA_ISO.2 == [CE_ISO | x = z and y != x]
   PA_ISO.3 == [CE_ISO | y = z and y != x]
   ```

   (c) Ordered types permutation:

   ```
   PERM_SCA.xyz == [CE_SCA | x < y < z]
   PERM_SCA.xzy == [CE_SCA | x < z < y]
   PERM_SCA.yxz == [CE_SCA | y < x < z]
   PERM_SCA.yzx == [CE_SCA | y < z < x]
   PERM_SCA.zyx == [CE_SCA | z < y < x]
   PERM_SCA.zxy == [CE_SCA | z < x < y]
   ```

   (d) Boundary value analysis: e.g. divide `x <= y` into cases `x < y` and `x=y`.

6. Test template hierarchies: draw for the simple triangle example so far. Note that if all children are disjoint and form the parent that: (1) all leaf nodes are disjoint; and (2) their disjunction/union is the VIS!
7. Test template: path from VIS to leaf; or expanded schema.
8. Prune infeasible paths. Explain and show an example of doing this with Alloy.
9. Generate inputs and expected outputs using Alloy. Show with `PERM_SCA.xyz` example.
10. Recap the process.
11. Discuss automation. The example is small but tedious. Tools/frameworks: Fastest (Flowgate consulting), Smartesting, SpecExplorer (Microsoft).

```
module Triangle

enum Triangle { Equilateral, Isoceles, Scalene, Invalid }

pred validTriangle[x, y, z : Int] {
  x < add[y, z] and y < add[x, z] and z < add[x, y]
}

pred validCase [x, y, z : Int, class : Triangle] {
  validTriangle[x, y, z]
  #(x+y+z) = 1 <=> class = Equilateral
  #(x+y+z) = 2 <=> class = Isoceles
  #(x+y+z) = 3 <=> class = Scalene
}

pred invalidCase [x, y, z : Int, class : Triangle] {
  not validTriangle[x, y, z]
  class = Invalid
}

pred Tri [x, y, z : Int, class : Triangle] {
  validCase[x, y, z, class] or invalidCase [x, y, z, class]
}

pred CE_EQU [x, y, z : Int, class : Triangle] {
  validCase [x, y, z, class]
  x = y and y = z
}

pred TT_PERM_SCA_xyz [x, y, z : Int] {
  validTriangle [x, y, z]
  #(x+y+z) = 3
  x < z and z < y
}

pred PERM_SCA_xyz [x, y, z : Int, class : Triangle] {
  Tri [x, y, z, class]
  TT_PERM_SCA_xyz [x, y, z]
}
run PERM_SCA_xyz for 6 Int

pred TB_INV_1_2 [x, y, z : Int] {
  not validTriangle [x, y, z]
  x >= add[y, z] and y >= add[x, z] and z >= add[x, y]
  x = 0 and y = 0 and z != 0
}
run TB_INV_1_2 for 6 Int expect 0

run CE_EQU for 6 Int
```