FSML → Scala

Test-Data Generation

Testing Frameworks

- ScalaCheck
- ScalaTest
- specs2

ScalaCheck

- automated property-based testing
- inspired by Haskell library QuickCheck

ScalaCheck

```
import org.scalacheck.Properties
import org.scalacheck.Prop.forAll

object StringSpecification extends Properties("String") {

 property("startsWith") = forAll { (a: String, b: String) =>
    (a+b).startsWith(a)
 }

 property("concatenate") = forAll { (a: String, b: String) =>
    (a+b).length > a.length && (a+b).length > b.length
 }

 property("substring") = forAll { (a: String, b: String, c: String) =>
    (a+b+c).substring(a.length, a.length + b.length) == b
 }

}
```

ScalaTest

- many testing styles available, e.g.
  - FunSuite
  - SetSpec
  - PropSpec
  - WordSpec
  - ...
- bindings for ScalaCheck

ScalaTest

SetSpec

```
import org.scalatest.Spec

class SetSpec extends Spec {

 object `A Set` {
    object `when empty` {
      def `should have size 0` {
        assert(Set.empty.size == 0)
      }
    }
  }

}
```

WordSpec

```
import org.scalatest.WordSpec

class SetSpec extends WordSpec {

 "A Set" when {
    "empty" should {
      "have size 0" in {
        assert(Set.empty.size == 0)
      }
    }
  }

}
```

specs2

- library for executable software specifications
  - unit specifications
  - acceptance specifications
- bindings for ScalaCheck

Expected Errors

- parser
  - unable to parse CS for certain inputs
- simulator
  - probably throws unexpected exceptions

Strategy for ParserTest

1. generate AST (could be semantically incorrect)
2. generate CS from AST with kiama (unparsing)
3. parse CS
4. compare parsed AST with generated AST

Strategy for SimulatorTest

1. generate valid AST with
    a. single initial state
    b. only reachable states
    c. …
2. generate valid input for AST
3. simulate FSM over AST with input