Teaching Concepts of Programming Languages in Scala with WebLab

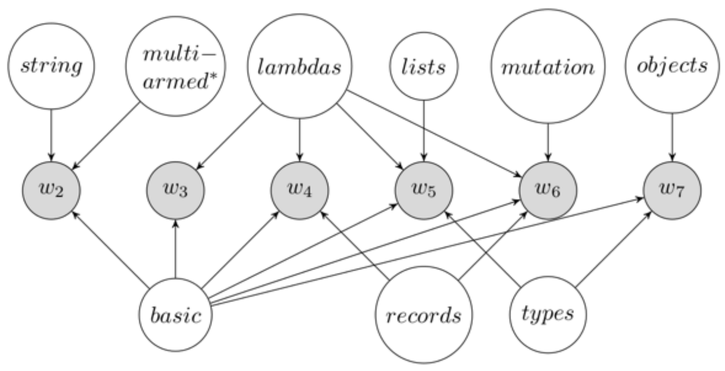
Teaching Approaches

There are various ways to teach programming language concepts.

- ☐ Study real world languages
- ☐ Study formal semantics
- ☑ Implement definitional interpreters*

Concepts of Programming Languages

Every week students learn new programming language concepts, and implement them in Scala as part of a Scheme-like language.



Reference Implementation

```
trait NumInterpreter extends Interpreter {
  abstract override def interp(e: ExprC) = e match {
    case NumC(n)     => NumV(n)
    case PlusC(l, r) => NumV(interp(l) + interp(r))
    case _           => super.interp(e)
  }
}

class Week2Interpreter extends BasicInterpreter
                          with NumInterpreter
                          with IfInterpreter
```

Our reference implementation uses Scala's mixin trait composition to override and extend each week's implementation, for high reuse and low maintainance.

Test Definition DSL

```
Cluster("not operator", List(
  Pos("(not true)"),
  Pos("(not false)"),
  Pos("(not (not true))"),
  Neg("(not 2)")
))
```

Our DSL in Scala allows us to generate new test cases efficiently.

Test Generator

The test generator generates a ScalaTest test suite from our test definition DSL, asserting that the student implementation produces the same results as our reference implementation.

Specification Tests

```
class TestSpec extends FunSuite {
  test("not operator") {
    expect(UnOpExt("not", TrueExt())) {
      parse("(not true)")
    }
    expect(BoolV(false)) {
      interp(desugar(UnOpExt("not", TrueExt())))
    }
  }
}
```

The generated specification test suite is used for grading, but students use it also to get immediate feedback on their progress.

Specification Meta-tests

```
test("Students should test for duplicate let-binding") {
  testParser(new Week3Interpreter {
    override def checkLetBinding(vars: List[String]) = {
      // Do not check for duplicates
    }
  })
}
```

To test the student's test suite, we override a single aspect of our reference implementation to be faulty, and verify that the student's test suite catches this.

WebLab

CONCEPTS OF PROGRAMMING LANGUAGES / ASSIGNMENT

1.0/4.0 | 1 2 3 4

Basic Interpreter ✔ Completed

Example

`(not true)`

`UnOpExt("not", TrueExt())`

`IfC(TrueC(), FalseC(), TrueC())`

`BoolV(false)`

Solution

```
def parse(sexpr: SExpr): ExprExt = sexpr match {
  case SNum(n)                          => NumExt(n)
  case SSym("true")                     => TrueExt()
  case SSym("false")                    => FalseExt()
  case SList(List(SSym("if"), c, t, e)) =>
    IfExt(parse(c), parse(t), parse(e))
  case SList(List(SSym("not"), e))      =>
    UnOpExt(sym, parse(e))
}

def desugar(e: ExprExt): ExprC = e match {
  case TrueExt()          => TrueC()
  case FalseExt()         => FalseC()
  case NumExt(n)          => NumC(n)
  case UnOpExt("not", e) =>
    IfC(desugar(e), FalseC(), TrueC())
}

def interp(e: ExprC) = e match {
  case TrueC()            => BoolV(true)
  case FalseC()           => BoolV(false)
  case NumC(n)            => NumV(n)
  case IfC(c, t, e)       => interp(c) match {
    BoolV(true)             => interp(t)
    BoolV(false)            => interp(e)
  }
}
```

Students implement their parser, desugarer and definitional interpreter in the WebLab learning management system.

Test

```
test("Nesting expressions") {
  expect(BoolV(false)) {
    interp(desugar(parse("(not (not true))")))
  }
}
test("Catch erroneous behavior") {
  intercept[InterpException] {
    interp(desugar(parse("(not 2)")))
  }
}
```

To verify their implementation students write their own unit tests.

Console | Discussion | Revision History

Saved | ▶ Your Test | ▶ Spec-test | Submit | Reset

```
Status: Done
Test score: 42/42
```

WebLab gives students immediate feedback about their code by running their unit tests or our hidden specification tests.

* Shriram Krishnamurthi, and Joe Gibbs Politz. "Programming and Programming Languages." Brown University (2015) http://papl.cs.brown.edu/2015/

Tim van der Lippe, Thomas Smith, Daniël Pelsmaeker, and Eelco Visser. "A Scalable Infrastructure for Teaching Concepts of Programming Languages in Scala with WebLab." Proceedings of the 2016 7th ACM SIGPLAN Symposium on Scala. ACM (2016)

Tim van der Lippe
T.J.vanderLippe@student.tudelft.nl

Thomas Smith
T.N.Smith@student.tudelft.nl

Daniël Pelsmaeker
D.A.A.Pelsmaeker@student.tudelft.nl

Eelco Visser
E.Visser@tudelft.nl

TU Delft