Software verification in Scala with Leon

Romain Ruetschi, EPFL

August 2015

What is software verification?

What is software verification?

Software verification aims at making software safer and more reliable.

It does so by verifying statically that a program confirms to a given specification and that it cannot crash at run-time.

What is Leon?

What is Leon?

Leon is a verification, repair and synthesis system for Scala.

What is Leon?

Leon takes as input a Scala source file, and generates individual verification conditions corresponding to different properties of the program.

It then tries to prove or disprove (by yielding a counter-example) that the verification conditions hold.

Verification

Verification

Pre- and post-conditions

```scala
def neg(x: Int): Int = {
  require(x >= 0)
  -x
} ensuring(res => res <= 0)
```

Verification

Array access safety

For each array variable, Leon carries along a symbolic information on its length.

This information is used to prove that each expression used as an index in the array is both positive and strictly smaller than its length.

Verification

Pattern matching exhaustiveness

Takes pre-conditions into account to verify that pattern matches are exhaustive.

```
def getHead(l: List): Int = {
  require(!l.isInstanceOf[Nil])
  l match {
    case Cons(x, _) => x
  }
}
```

Repair and synthesis

Repair and synthesis

Leon can automatically repair your program if it doesn't satisify its specification.

Moreover, it can also synthesize code from a specification!

Demo

Under the hood

Under the hood

Leon is itself written in Scala

Under the hood

It makes use of the Scala compiler to parse input files and typecheck programs.

Under the hood

Then the Scala AST is converted to a PureScala AST.

Under the hood

A lot of black magic (to me) happens.

Under the hood

Most of the hard work required to prove or disprove various properties of the program is delegated to a SMT solver.

SMT solver

SMT stands for Satisfiability Modulo Theories: First-order logic formulas over various *theories* such as real numbers, integers, lists, arrays, ADTs, and others.

A SMT solver tries to either prove that a given formula holds, or yields a counter-example.

SMT solver

Leon can make use of different SMT solvers, such as Z3 or CVC4, thanks to the SMT-Lib standard.

Bachelor semester project

Bachelor semester project

An encoding of Any for Leon

An encoding of Any for Leon

Adds support for uni-typed programs such as

```
def reverse(lst: Any): Any = {
  if (lst == Nil()) Nil()
  else reverse(lst.tail) ++ Cons(lst.head, Nil())
} ensuring (_.contents == lst.contents)

def reverseReverseIsIdentity(lst: Any) = {
  reverse(reverse(lst)) == lst
}.holds
```

An encoding of Any for Leon

Mostly an experiment, as using Any is generally frowned upon in the Scala community.

Has nonetheless interesting applications, such as eg. automatically porting theorems from Lisp-based theorem provers like ACL2.

An encoding of Any for Leon

Nothing too fancy. It's just a pre-processing phase, that encodes Any as a sum type and lifts expressions into it.

Allowed us to add support for Any without touching the rest of the system.

An encoding of Any for Leon

Before

```
case class Box(value: Int)

def double(x: Any): Any = x match {
    case n: Int => n * 2
    case Box(n) => Box(n * 2)
    case _      => x
}

double(42)
```

An encoding of Any for Leon

After

```
sealed abstract class Any1
case class Any1Int(value: Int) extends Any1
case class Any1Box(value: Box) extends Any1

def double(x: Any1): Any1 = x match {
    case Any1Int(n)       => Any1Int(n * 2)
    case Any1Box(Box(n)) => Any1Box(Box(n * 2))
    case _                => x
}

double(Any1Int(42))
```

Resources

Resources

http://leon.epfl.ch

http://leon.epfl.ch/doc

http://lara.epfl.ch/w/leon

https://github.com/epfl-lara/leon

Thank you!

Thank you!

If you have any questions or just want to get in touch, I am @_romac on Twitter.