Experiments in Formal Verification of Scala Code

Ramon Boss, Anna Doukmak

2019-06-14

Table of Contents

Table of Contents

Formal Verification

Problems Verifying Bitcoin-S

Conclusion

Stainless

- Pure Scala
- Pre- and Postcondition
- Outcome: valid, invalid, unknown

Example: max

```scala
def max(x: Int, y: Int): Int = {
  val d = x - y
  if (d > 0) x
  else y
}
```

https://epfl-lara.github.io/stainless/tutorial.html#warm-up-max

Example: max

```scala
def max(x: Int, y: Int): Int = {
  val d = x - y
  if (d > 0) x
  else y
} ensuring (res =>
  x <= res && y <= res && (res == x || res == y))
```

https://epfl-lara.github.io/stainless/tutorial.html#warm-up-max

Example: max

```
[...     ]
[  Info  ]  - Result for ’postcondition’ VC for max @2:3:
[Warning ]  => INVALID
[Warning ] Found counter-example:
[Warning ]   x: Int -> -2147483648
[Warning ]   y: Int -> 2147483647
[...     ]
```

Example: max

```scala
def max(x: Int, y: Int): Int = {
  require(0 <= x && 0 <= y)
  val d = x - y
  if (d > 0) x
  else y
} ensuring (res =>
  x <= res && y <= res && (res == x || res == y))
```

https://epfl-lara.github.io/stainless/tutorial.html#warm-up-max

Properties

- Bitcoin-S
- No-Inflation Property
- Addition-with-Zero Property

Table of Contents

Formal Verification

Problems Verifying Bitcoin-S

Conclusion

Turning Object into Case Object

This:

```scala
object Satoshis extends BaseNumbers[Satoshis] {
  val zero = Satoshis(Int64.zero)
  val one = Satoshis(Int64.one)
}
```

Becomes this:

```scala
case object Satoshis extends BaseNumbers[Satoshis] {
  val zero = Satoshis(Int64.zero)
  val one = Satoshis(Int64.one)
}
```

Getting Rid of Abstract Type Member

This:

```scala
sealed abstract class CurrencyUnit {
  type A

  protected def underlying: A
}

sealed abstract class Satoshis extends CurrencyUnit {
  override type A = Int64
}
```

Becomes this:

```scala
sealed abstract class CurrencyUnit {
  protected def underlying: Int64
}

sealed abstract class Satoshis extends CurrencyUnit
```

BigInt &-Function to Bound Check

This:

```scala
sealed abstract class Number {
  def andMask: BigInt
  def checkResult(result: BigInt): BigInt = {
    require((result & andMask) == result)
    result
  }
}
```

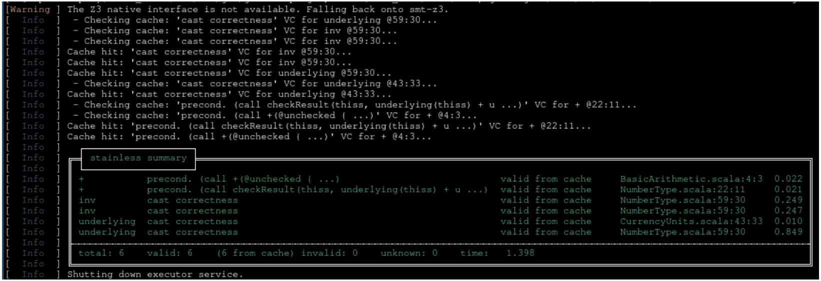
Becomes this:

```scala
sealed abstract class Number {
  def checkResult(result: BigInt): BigInt = {
    require(
      result <= BigInt("9223372036854775807") &&
      result >= BigInt("-9223372036854775808")
    )
    result
  }
}
```

Formal Specification

```scala
def +(c: CurrencyUnit): CurrencyUnit = {
  require(c.satoshis == Satoshis.zero)
  Satoshis(
    satoshis.underlying + c.satoshis.underlying
  )
} ensuring(res => res.satoshis == this.satoshis)
```

Output of Stainless



```
[Warning ] The Z3 native interface is not available. Falling back onto smt-z3.
[  Info  ]  - Checking cache: 'cast correctness' VC for underlying @59:30...
[  Info  ]  - Checking cache: 'cast correctness' VC for inv @59:30...
[  Info  ]  - Checking cache: 'cast correctness' VC for inv @59:30...
[  Info  ] Cache hit: 'cast correctness' VC for inv @59:30...
[  Info  ] Cache hit: 'cast correctness' VC for inv @59:30...
[  Info  ] Cache hit: 'cast correctness' VC for underlying @59:30...
[  Info  ]  - Checking cache: 'cast correctness' VC for underlying @43:33...
[  Info  ] Cache hit: 'cast correctness' VC for underlying @43:33...
[  Info  ]  - Checking cache: 'precond. (call checkResult(thiss, underlying(thiss) + u ...)' VC for + @22:11...
[  Info  ]  - Checking cache: 'precond. (call +(@unchecked ( ...)' VC for + @4:3...
[  Info  ] Cache hit: 'precond. (call checkResult(thiss, underlying(thiss) + u ...)' VC for + @22:11...
[  Info  ] Cache hit: 'precond. (call +(@unchecked ( ...)' VC for + @4:3...
[  Info  ]
[  Info  ]   stainless summary
[  Info  ]
[  Info  ]  +           precond. (call +(@unchecked ( ...)                            valid from cache     BasicArithmetic.scala:4:3   0.022
[  Info  ]  +           precond. (call checkResult(thiss, underlying(thiss) + u ...)  valid from cache     NumberType.scala:22:11      0.021
[  Info  ]  inv         cast correctness                                              valid from cache     NumberType.scala:59:30      0.249
[  Info  ]  inv         cast correctness                                              valid from cache     NumberType.scala:59:30      0.247
[  Info  ]  underlying  cast correctness                                              valid from cache     CurrencyUnits.scala:43:33   0.010
[  Info  ]  underlying  cast correctness                                              valid from cache     NumberType.scala:59:30      0.849
[  Info  ]
[  Info  ]  total: 6    valid: 6    (6 from cache) invalid: 0    unknown: 0    time:   1.398
[  Info  ]
[  Info  ] Shutting down executor service.
```

Table of Contents

Formal Verification

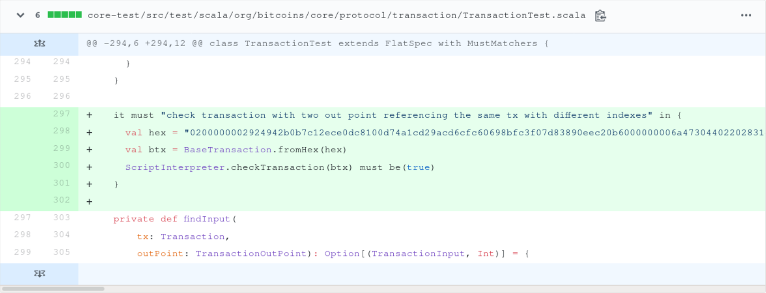
Problems Verifying Bitcoin-S

Conclusion

Conclusion

- Write verifiable code
- We verified not original Bitcoin-S code
- Provided feedback to Stainless developers
- Found a bug in Bitcoin-S

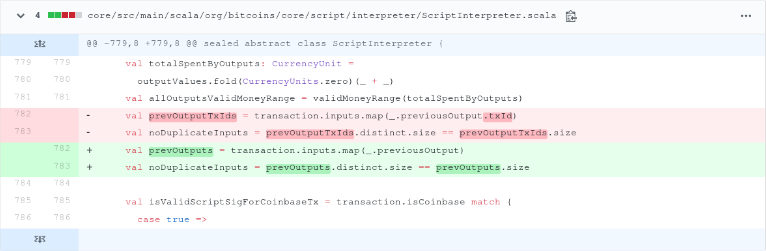
Found Bug in Bitcoin-S



```
6 core-test/src/test/scala/org/bitcoins/core/protocol/transaction/TransactionTest.scala

@@ -294,6 +294,12 @@ class TransactionTest extends FlatSpec with MustMatchers {
        }
      }

+   it must "check transaction with two out point referencing the same tx with different indexes" in {
+     val hex = "0200000002924942b0b7c12ece0dc8100d74a1cd29acd6cfc60698bfc3f07d83890eec20b6000000006a47304402202831
+     val btx = BaseTransaction.fromHex(hex)
+     ScriptInterpreter.checkTransaction(btx) must be(true)
+   }
+
    private def findInput(
        tx: Transaction,
        outPoint: TransactionOutPoint): Option[(TransactionInput, Int)] = {
```



```
4 core/src/main/scala/org/bitcoins/core/script/interpreter/ScriptInterpreter.scala

@@ -779,8 +779,8 @@ sealed abstract class ScriptInterpreter {
      val totalSpentByOutputs: CurrencyUnit =
        outputValues.fold(CurrencyUnits.zero)(_ + _)
      val allOutputsValidMoneyRange = validMoneyRange(totalSpentByOutputs)
-     val prevOutputTxIds = transaction.inputs.map(_.previousOutput.txId)
-     val noDuplicateInputs = prevOutputTxIds.distinct.size == prevOutputTxIds.size
+     val prevOutputs = transaction.inputs.map(_.previousOutput)
+     val noDuplicateInputs = prevOutputs.distinct.size == prevOutputs.size

      val isValidScriptSigForCoinbaseTx = transaction.isCoinbase match {
        case true =>
```