Introducing Reflection into a Verification System

Julie Giunta

School of Computer and Communication Sciences
Semester Project Report

Supervisor
Romain Edelmann
EPFL / LARA

Supervisor
Prof. Viktor Kunčak
EPFL / LARA

June 2019

Introducing Reflection into a Verification System

Julie Giunta
EPFL, Switzerland
julie.giunta@epfl.ch

Abstract

Stainless is a tool for verifying Scala programs. During verification, tree reflection is sometimes needed. It can be used to send constraints to an underlying constraint solving during program execution. With tree reflection, the program can be described as an algebraic data type. You can then type check and interpret it within Stainless.

1 Introduction

The goal of this project was to implement tree reflection within Stainless. Tree reflection means to allow access to an expression as an algebraic data type. For example, if you had this expression:

```
x + y * z
```

the corresponding algebraic data type could be:

```
Plus(Var("x"), Times(Var("y"), Var("z")))
```

For this purpose, we defined which expressions we wanted to be able to reflect and how to describe them as algebraic data types. Furthermore, we defined some basic types to support type checking on our expressions. As the underlying expressions could be interpreted, we added an interpreter for our algebraic data types that returned the same result but wrapped in its reflection.

2 Implementation

The system is written in Pure Scala, thus allowing Stainless to verify it. As Stainless ensures lot of properties, the code has to be as simple as it could be, otherwise Stainless is not able to terminate the verification in a sustainable time.

2.1 Expressions

Expressions are algebraic data types. They represent the abstract syntax tree (AST) of a program that can be type checked and interpreted. It was implemented as an abstract class, *Expr*, and case classes that extend *Expr*.

The expressions that will be find in the leaves of the AST are the literals. They are the "basic unit" of expressions. For example, the literal for a character is defined as:

```
case class CharLiteral(value: Char) extends Expr
```

There are also literals that represent integers, booleans, strings and fractions. They are defined in a similar manner except for fractions which take a tuple of BigInt, the first element of the tuple representing the numerator and the second one the denominator. They all correspond to one of the basic types described in the following section.

Another type of value (leaf of the AST) is lambda definition:

```
case class Lambda(params: List[(Identifier, Type)],
                  body: Expr) extends Expr
```

It represents lambda-expressions such as

```
λ x: BigInt, λ y: BigInt → x + y
```

In the program, a list containing *(x, BigInt)* and *(y, BigInt)* represents the *params*. In fact, x and y would be Identifiers. Identifiers are defined as a case class that takes a string as parameter, the "name" of what needs to be identified. $x + y$ represents the *body*. In the body, x and y would be represented by *Variable*, which is an expression defined by:

```
case class Variable(id : Identifier) extends Expr
```

The nodes of the AST represents operations done on the leaves and on other nodes. For example, the addition operation is implemented as:

```
case class Plus(lhs: Expr, rhs: Expr) extends Expr
```

Here, *Plus* is a node in the AST and *lhs* and *rhs* are the children of this node. The other arithmetic operations (minus, unary minus, times, division, remainder and modulo) are defined in a similar way, except for the unary minus which takes only one argument.

In addition of arithmetic, there are also expressions to represent:

- String operations such as concatenation, computing the length and taking a subset of the given string expression
- Equality test on two expressions
- Comparisons on integers, fractions and characters expressions such as greater than ($>$), greater equals ($\geq$), less than ($<$) and less equals ($\leq$).
- Logical operations on booleans expressions such as logical and, logical or, logical implication and logical not
- If-then-else expression

- Let expression, an expression which gives a certain value to an identifier and then uses this definition in the subsequent expression

Another type of expression is *Application*:

```
case class Application(callee: Expr, args: List[Expr])
    extends Expr
```

It is used when we want to apply arguments to a *Lambda*. For example, if our lambda is

```
add = λ x: BigInt, λ y: BigInt → x + y
```

then $add(1, 2)$ would correspond to an *Application* where *callee* is *add* and a list containing 1 and 2 represents *args*.

When there is a problem during the interpretation, the program outputs an expression which represents an error:

```
case class ErrorValue(error: String) extends Expr
```

2.2 Types

Types are represented as an abstract class *Type* and case classes that extend it and represent basic types of Stainless. There is a type for each kind of literal. For example, the type corresponding to a character is:

```
case class CharType() extends Type
```

In the same way, *IntegerType* is for integers, *BooleanType* for booleans, *StringType* for strings and *RealType* for fractions.

The type that corresponds to lambdas is:

```
case class FunctionType(from: List[Type], to: Type) extends Type
```

where *from* contains the types of the parameters *params* of *Lambda* and *to* represents the type of the body of *Lambda*.

2.3 DSL

To be able to define expressions in a more convenient way, the user can call some methods of the Domain Specific Language (DSL).

For literals, the methods are named with the first letter of the literal in capital. For example, for a *CharLiteral*:

```
def C(c : Char) = CharLiteral(c)
```

For expressions that need an Identifier as argument, the user only has to provide a string which represents the name of this Identifier. For example, to define a *Let* expression:

```
def let(name: String, tpe: Type, value: Expr)
     (body: Identifier => Expr): Expr = {
                   val id = Identifier(name)
                   Let(id, tpe, value, body(id))
          }
```

For arithmetic, string concatenation, comparisons and logical operators, the methods are defined in the abstract class *Expr* using operator overloading. For example, for *Plus*:

```
def +(rhs: Expr) = Plus(this, rhs)
```

Then, if you need to express the addition of 1 and 2, you can write it as:

```
I(1) + I(2)
```

and it will represent

```
Plus(I(1), I(2))
```

Unary operators such as *UMinus* and *Not* were defined by:

```
def unary_- = UMinus(this)
def unary_! = Not(this)
```

Then, to express -1, the user only needs to write:

```
-I(1)
```

and to express the negation of the boolean true:

```
! B(true)
```

2.4 Type checker

The type checker was implemented in a recursive manner. It is a method that takes as arguments an expression and an environment which is represented by a Map of Identifier to Type. It does a pattern match on the given expression and applies an inference rule to obtain either *None()* if the expression does not type check or *Some(resulting type)* if the expression type checks.

The inference rules are (E representing the environment):

For literals:

$$\text{CharLiteral}\ \frac{c\ is\ a\ Char}{E \vdash CharLiteral(c) : CharType}$$

$$\text{IntegerLiteral}\ \frac{i\ is\ a\ BigInt}{E \vdash IntegerLiteral(i) : IntegerType}$$

$$\text{BooleanLiteral}\ \frac{b\ is\ a\ Boolean}{E \vdash BooleanLiteral(b) : BooleanType}$$

$$\text{StringLiteral}\ \frac{s\ is\ a\ String}{E \vdash StringLiteral(s) : StringType}$$

$$\text{FractionLiteral}\ \frac{n\ is\ a\ BigInt,\ d\ is\ a\ BigInt}{E \vdash FractionLiteral((n,d)) : RealType}$$

$$\text{Variable}\ \frac{v : T \in E}{E \vdash v : T}$$

$$\text{Let}\ \frac{E \vdash value : T1 \quad E, id : T1 \vdash body : T2}{E \vdash Let(id, T1, value, body) : T2}$$

$$\text{If-then-else}\ \frac{E \vdash cond : BooleanType \quad E \vdash thenn : T \quad E \vdash elze : T}{E \vdash IfExpr(cond, thenn, elze) : T}$$

For lambdas:

$$\text{Lambda}\ \frac{E_{[id1:=t1,...,idN:=tN]} \vdash body : T}{E \vdash Lambda([(id_1, t_1), ..., (id_N, t_N)], body) : FunctionType([t_1, ..., t_N], T)}$$

$$\text{Application}\ \frac{E \vdash callee : FunctionType([t_1, ..., t_N], T)\ E \vdash a_1 : t_1\ ...\ E \vdash a_N : t_N}{E \vdash Application(callee, [a_1, ..., a_N]) : T}$$

For arithmetic expressions:

$$\text{General arithmetic}\ \frac{E \vdash lhs : T\ E \vdash rhs : T}{E \vdash op(lhs, rhs) : T}\ \text{Unary minus}\ \frac{E \vdash e : T}{E \vdash UMinus(e) : T}$$

The general arithmetic rule is applied by *Plus, Minus, Times, Division, Modulo* and *Remainder* where *op* represents the corresponding operation. For the last two, T can only be *IntegerType*. For the others, T is either *IntegerType* or either *RealType*.
The Unary minus rule is applied by *UMinus* and T can be *IntegerType* or *RealType*.

For string operations:

$$\text{Strings concatenation}\ \frac{E \vdash lhs : StringType\ E \vdash rhs : StringType}{E \vdash StringConcat(lhs, rhs) : StringType}$$

$$\text{Substring}\ \frac{E \vdash e : StringType\ E \vdash start : IntegerType\ E \vdash end : IntegerType}{E \vdash SubString(e, start, end) : IntegerType}$$

$$\text{String length}\ \frac{E \vdash e : StringType}{E \vdash StringLength(e) : IntegerType}$$

For comparisons expressions:

$$\text{Equals}\ \frac{E \vdash lhs : T1\ E \vdash rhs : T2}{E \vdash Equals(lhs, rhs) : BooleanType}$$

$$\text{Other comparisons}\ \frac{E \vdash lhs : T \quad E \vdash rhs : T}{E \vdash op(lhs, rhs) : T}$$

For equals, *T1* and *T2* can be of any type but *lhs* and *rhs* must type check. For other comparisons, *T* must be either *IntegerType, RealType* or *CharType*. This rule is applied by *LessThan, GreaterThan, LessEquals* and *GreaterEquals*.

For logical operations:

$$\text{Logical operators}\ \frac{E \vdash lhs : BooleanType \quad E \vdash rhs : BooleanType}{E \vdash op(lhs, rhs) : BooleanType}$$

$$\text{Not}\ \frac{E \vdash e : BooleanType}{E \vdash Not(e) : BooleanType}$$

For logical operators, *op* can be *And, Or* or *Implies*.

2.5 Interpreter

The interpreter is a small-step interpreter. It has two main methods: *interpret*, which takes an expression and returns the fully interpreted expression (either a literal, a lambda or an error value) and *next*, which takes an expression and returns an option, either *Some(the next small-step of interpretation of the expression)* or either *None()* if it is stuck. Being stuck can happen for two reasons. First, the expression is already fully evaluated, thus we cannot progress. Second, the expression does not make sense, thus there are no inference rule to progress in the interpretation.

The interpreter substitutes the variables in the rest of the expression as soon as their value is defined. For example, if you have this pseudo-code below:

```
let x = 1; let x = x + 2; x + 4
```

substitution of the first x will return:

```
let x = 1; let x = 1 + 2; x + 4
```

because the second x masks the first one in the $x + 4$ statement. Then, after the second x has been evaluated, the substitution would return:

```
let x = 1; let x = 3; 3 + 4
```

The *next* method follows these inference rules, where $\rightarrow$ means a step of evaluation and v represents a fully interpreted expression (a literal, a lambda or an error):

For Let:

$$\frac{e \rightarrow e'}{Let(id, t, e, body) \rightarrow Let(id, t, e', body)}$$

$$\frac{v\ is\ a\ value}{Let(id, t, v, body) \rightarrow body_{[id:=v]}}$$

For Application:

$$\frac{callee \to callee'}{Application(callee, args) \to Application(callee', args)}$$

$$\frac{c \ is \ a \ Lambda, \ e_i \to e'_i}{Application(c, [v_1, ..., v_{i-1}, e_i, e_{i+1}, ..., e_N]) \to Application(c, [v_1, ..., v_{i-1}, e'_i, e_{i+1}, ..., e_N])}$$

$$\frac{c \ is \ a \ Lambda([(id_1, t_1), ..., (id_N, t_N)], body), \ v_1, ..., v_N \ are \ values}{Application(c, [v_1, ..., v_N]) \to body_{[id1:=v1,...,idN:=vN]}}$$

For if-then-else expressions:

$$\frac{cond \to cond'}{IfExpr(cond, thenn, elze) \to Let(cond', thenn, elze)}$$

$$\frac{v \ is \ BooleanLiteral(true)}{IfExpr(v, thenn, elze) \to thenn} \qquad \frac{v \ is \ BooleanLiteral(false)}{IfExpr(v, thenn, elze) \to elze}$$

For arithmetic expressions:

$$\text{Arithmetic 1} \ \frac{lhs \to lhs'}{Expr(lhs, rhs) \to Expr(lhs', rhs)}$$

$$\text{Arithmetic 2} \ \frac{v \ is \ an \ IntegerLiteral, \ rhs \to rhs'}{Expr(v, rhs) \to Expr(v, rhs')}$$

$$\text{Arithmetic 3} \ \frac{v \ is \ a \ FractionLiteral, \ rhs \to rhs'}{Expr(v, rhs) \to Expr(v, rhs')}$$

$$\text{Arithmetic 4} \ \frac{v1, \ v2 \ are \ IntegerLiterals}{Expr(v1, v2) \to v1 \ op \ v2}$$

$$\text{Arithmetic 5} \ \frac{v1, \ v2 \ are \ FractionLiterals}{Expr(v1, v2) \to v1 \ op \ v2}$$

$$\frac{e \to e'}{UMinus(e) \to UMinus(e')} \qquad \frac{v \ is \ an \ IntegerLiteral \ or \ a \ FractionLiteral}{UMinus(v) \to -v}$$

The rules Arithmetic 1, 2, 3, 4 and 5 are used by the *Expr*: Plus, Minus, Times, Division and their corresponding operation *op*: +, -, *, / The rules Arithmetic 1, 2 and 4 are implemented by the *Expr*: Remainder, Modulo and their corresponding operation *op*: %, mod.

For operations on strings:

$$\frac{lhs \to lhs'}{StringConcat(lhs, rhs) \to StringConcat(lhs', rhs)}$$

$$\frac{v \ is \ a \ StringLiteral, \ rhs \to rhs'}{StringConcat(v, rhs) \to StringConcat(v, rhs')}$$

$$\frac{v1,\ v2\ are\ StringsLiterals}{StringConcat(v1, v2) \rightarrow v1 ++ v2}$$

$$\frac{e \rightarrow e'}{SubString(e, start, end) \rightarrow SubString(e', start, end)}$$

$$\frac{v\ is\ a\ StringLiteral,\ start \rightarrow start'}{SubString(v, start, end) \rightarrow SubString(v, start', end)}$$

$$\frac{v1\ is\ a\ StringLiteral,\ v2\ is\ an\ IntegerLiteral,\ end \rightarrow end'}{SubString(v1, v2, end) \rightarrow SubString(v1, v2, end')}$$

$$\frac{v1\ is\ a\ StringLiteral,\ v2, v3\ are\ IntegerLiterals}{SubString(v1, v2, v3) \rightarrow v1.bigSubstring(v2, v3)}$$

$$\frac{e \rightarrow e'}{StringLength(e) \rightarrow StringLength(e')} \quad \frac{v\ is\ a\ StringLiteral}{StringLength(v) \rightarrow v.bigLength}$$

For comparisons:

$$\frac{lhs \rightarrow lhs'}{Expr(lhs, rhs) \rightarrow Expr(lhs', rhs)}$$

$$\frac{v\ is\ a\ lit,\ rhs \rightarrow rhs'}{Expr(v, rhs) \rightarrow Expr(v, rhs')} \quad \frac{v1,\ v2\ are\ lits}{Expr(v1, v2) \rightarrow v1\ op\ v2}$$

For the *Expr* Equals, *v1* and *v2* can be any type of value, *v1* and *v2* can even be different type of literals and then *op* is $==$. For the *Expr* LessThan, LessEquals, GreaterThan, GreaterEquals, *lit* are IntegerLiteral, FractionLiteral and CharLiteral. *v1* and *v2* must be the same kind of literals. The corresponding *op* are: $<, \leq, >, \geq$.

For logical operators:

$$\text{General logic}\ \frac{lhs \rightarrow lhs'}{Expr(lhs, rhs) \rightarrow Expr(lhs', rhs)}$$

$$\text{And and Implies 1}\ \frac{v\ is\ BooleanLiteral(true),\ rhs \rightarrow rhs'}{Expr(v, rhs) \rightarrow rhs'}$$

$$\text{And and Implies 2}\ \frac{v1\ is\ BooleanLiteral(true), v2\ is\ a\ BooleanLiteral}{Expr(v1, v2) \rightarrow v2}$$

$$\frac{v\ is\ a\ BooleanLiteral(false),\ rhs \rightarrow rhs'}{Or(v, rhs) \rightarrow rhs'}$$

$$\frac{v1\ is\ a\ BooleanLiteral(false), v2\ is\ a\ BooleanLiteral}{Or(v, v2) \rightarrow v2}$$

$$\frac{v\ is\ a\ BooleanLiteral(false)}{And(v, rhs) \rightarrow BooleanLiteral(false)}$$

$$\frac{v\ is\ a\ BooleanLiteral(true)}{Or(v, rhs) \rightarrow BooleanLiteral(true)}$$

$$\frac{v \text{ is a } BooleanLiteral(false)}{Implies(v, rhs) \to BooleanLiteral(true)}$$

$$\frac{e \to e'}{Not(e) \to Not(e')} \quad \frac{v \text{ is a } BooleanLiteral}{Not(v) \to !v}$$

General logic is used by And, Or and Implies.

2.6 Soundness theorem

The soundness theorem can be stated as: "If a program type checks, its evaluation does not get stuck"(1). To show that the system (or at least a subset of the Expressions) was a sound system, we proved with Stainless two lemmas, progress and preservation, on some of the Expressions.

2.6.1 Progress

Progress can be stated as: "If a program type checks, it is not stuck"(1). In the program, it is translated by a method which takes as arguments an expression *expr* and a type *t*. It has a precondition:

```
require(!isValue(expr) &&
    typecheck(expr, Map[Identifier, Type]()) == Some(t))
```

and a post condition:

```
next(expr).nonEmpty
```

where the *isValue* method returns true if *expr* is a literal, a *Lambda* or an *ErrorValue*. To ensure the post condition, we used the method *holds* of Stainless.

Stainless was not able to prove this as it was stated but fortunately, by using additional lemmas (*check* method of Stainless), Stainless did verify progress on a subset of expressions. The lemmas were added depending on the given expression. To do so, we used pattern matching. For expressions such as general arithmetic, string operations, comparisons, logical operators and if-then-else, we added an equality test on the type of the expression and checked that the fields of each expression had a consistent type. Furthermore, we pattern matched on the fields and checked the progress of the first non-evaluated field. For example, here are the added lemmas for *Plus*:

```
case Plus(lhs, rhs) => {
  check((t == IntegerType() || t == RealType()) &&
    typecheck(lhs, Map[Identifier, Type]()) == Some(t) &&
    typecheck(rhs, Map[Identifier, Type]()) == Some(t))
  (lhs, rhs) match{
        case (IntegerLiteral(_), IntegerLiteral(_)) => true
        case (FractionLiteral(_), FractionLiteral(_)) => true
        case (IntegerLiteral(_), _) => check(progress(rhs, t))
        case (FractionLiteral(_), _) => check(progress(rhs, t))
        case (_, _) => check(progress(lhs, t))
```

```
  }
}
```

For *Let*, we added:

```
case Let(id, tValue, value, body) =>
  value match{
    case _ if(isValue(value)) => true
    case _ => check(progress(value, tValue))
  }
```

For *Application*, Stainless could not verify it if it had an unbounded number of parameters, but we were able to prove progress on an *Application* which had only one or two parameters. To be able to do so, we defined:

```
case class FunctionType1(from: Type, to: Type) extends Type
case class FunctionType2(from1: Type, from2: Type , to: Type)
    extends Type

case class Lambda1(id: Identifier, t: Type, body: Expr)
    extends Expr
case class Lambda2(id1: Identifier, t1: Type,
    id2: Identifier, t2: Type, body: Expr) extends Expr

case class Application1(callee: Expr, arg: Expr) extends Expr
case class Application2(callee: Expr, arg1: Expr, arg2: Expr)
    extends Expr
```

Lambda1, Lambda2 and *Application1, Application2* follow the same rules for type checking and interpretation as *Lambda* and *Application* but with respectively one and two parameters. The lemmas that were added to prove progress on *Application1* are:

```
case Application1(callee, arg) =>
  callee match {
        case Lambda1(id, t, body) => arg match{
            case _ if(isValue(arg)) => true
            case _ => check(progress(arg, t))
        }
        case _ =>{
            val tArg = typecheck(arg,
              Map[Identifier, Type]())
            check(tArg.nonEmpty &&
              progress(callee, FunctionType1(tArg.get, t)))
        }
}
```

It is similar for *Application2*, except it checks progress on the first non-evaluated *arg* and type checks both *args*.

Due to the duration of verifying progress with Stainless, we could only verify together:

- Arithmetic, string operations and if-then-else expressions (approximately 4 hours for the post condition of progress only and 5 hours to prove all the others lemmas in progress)
- Let expressions, comparisons and logical operators (approximately 4 hours for the post condition of progress only and 5 hours to prove all the other lemmas in progress)
- Application1 (less than a minute to prove progress)
- Application2 (less than a minute to prove progress)

The duration may seem extremely long but Stainless had to control the exhaustiveness of every pattern matching and that there were no divisions, remainders or modulos by zero. It also verified every clause in each *check* method and then that the overall lemma was correct. Furthermore, it had to verify the post-condition of progress, which took the longest time. The output of Stainless for each verification is shown in the annexes.

2.6.2 Preservation

Preservation can be stated as: "If a program type checks and makes one [next] step [with the Interpreter], then the result again type checks"(1). In the program, it is translated by a method which takes as arguments an expression *e1* and a type *t*. It has a precondition:

```
require(typecheck(e1, Map[Identifier, Type]()) == Some(t) &&
    next(e1).nonEmpty)
```

and a post condition:

```
typecheck(e2, Map[Identifier, Type]()) == Some(t)
```

where *e2* represents *next(e1)*. To ensure the post condition, we used the method *holds* of Stainless.

Unforunately, as it is, Stainless is not able to prove preservation on any of the expressions (or at least not in less than ten hours). We tried to add lemmas with the *check* method of Stainless but due to the opacity of the verification system, it is too complicated to find which lemmas can improve the verification, or at least not worsen it.

2.7 Tests

We made tests to check if the type checker and the interpreter had a normal behaviour. By side effect, it also tested the DSL. Some of the tests were made using the method *holds* of Stainless. For example:

```
def testInterpretPlusInteger(): Boolean = {
                    interpret(I(1) + I(2)) == I(3)
          }.holds
```

Due to the duration of the verification, beside these tests, we used the ScalaTest library with the FunSuite class.

3 Conclusion

This project was challenging due to the opacity and duration of the verification in Stainless. Sometimes, I had to run Stainless for more than 8 hours to receive a result. The system did not give enough feedback to know if and where it was stuck and what could help it progress. Without the help of my supervisor, Romain Edelmann, I would not have found some of the tricks to help Stainless do the verification, like using pattern matching on list instead of if-then-else expressions to verify the ADT invariant of recursive functions.

I have also lost a lot of time creating a system too complex for Stainless to verify, groping toward the goal and following wrong paths.

Despite these complications, this project was a thrilling experience. It taught me to tackle a problem step-by-step and to use a verification system, which existence I was not aware of.

References

[1] LARA, EPFL, *Computer Language Processing, Lecture 9, CS-320, Edition 2018* in http://lara.epfl.ch/cc18:top, Date of access: 06.06.19.

Annexes

```
stainless summary

areBothBoolean          match exhaustiveness                                         valid   nativez3   src/main/scala/Typechecker.scala:145:17  0.020
areBothInt              match exhaustiveness                                         valid   nativez3   src/main/scala/Typechecker.scala:161:17  0.020
areBothIntOrReal        match exhaustiveness                                         valid   nativez3   src/main/scala/Typechecker.scala:178:17  0.017
areBothIntOrRealOrChar  match exhaustiveness                                         valid   nativez3   src/main/scala/Typechecker.scala:196:17  0.022
interpret               match exhaustiveness                                         valid   nativez3   src/main/scala/Interpreter.scala:39:25   0.322
isValue                 match exhaustiveness                                         valid   nativez3   src/main/scala/Interpreter.scala:16:17   0.019
next                    match exhaustiveness                                         valid   nativez3   src/main/scala/Interpreter.scala:56:17   0.223
next                    match exhaustiveness                                         valid   nativez3   src/main/scala/Interpreter.scala:62:59   0.226
next                    match exhaustiveness                                         valid   nativez3   src/main/scala/Interpreter.scala:63:62   0.224
progress                postcondition                                                valid   nativez3   src/main/scala/Progress.scala:26:9       0.897
progress                match exhaustiveness                                         valid   nativez3   src/main/scala/Progress.scala:28:17      0.571
progress                match exhaustiveness                                         valid   nativez3   src/main/scala/Progress.scala:38:33      0.107
progress                match exhaustiveness                                         valid   nativez3   src/main/scala/Progress.scala:39:70      0.051
progress                precond. (call check(progress(arg, t)))                      valid   nativez3   src/main/scala/Progress.scala:41:59      0.563
progress                precond. (call progress(arg, t))                             valid   nativez3   src/main/scala/Progress.scala:41:65      0.999
progress                precond. (call check(res))                                   valid   nativez3   src/main/scala/Progress.scala:45:49      0.857
progress                precond. (call progress(callee, FunctionType1(get[Type] ...) valid   nativez3   src/main/scala/Progress.scala:45:72      0.725
progress                precond. (call get[Type](tArg))                              valid   nativez3   src/main/scala/Progress.scala:45:103     0.572
subst                   match exhaustiveness                                         valid   nativez3   src/main/scala/Interpreter.scala:352:17  0.019
typecheck               match exhaustiveness                                         valid   nativez3   src/main/scala/Typechecker.scala:20:17   0.024
typecheck               match exhaustiveness                                         valid   nativez3   src/main/scala/Typechecker.scala:58:33   0.044
-------------------------------------------------------------------------------------------------------------------------------------------------------
total: 21   valid: 21   (0 from cache) invalid: 0    unknown: 0    time:   6.522
```

Figure 1: Verification of progress on Lambda1 and Application1

```
stainless summary

areBothBoolean          match exhaustiveness                                         valid   nativez3   src/main/scala/Typechecker.scala:145:17  0.158
areBothInt              match exhaustiveness                                         valid   nativez3   src/main/scala/Typechecker.scala:161:17  0.020
areBothIntOrReal        match exhaustiveness                                         valid   nativez3   src/main/scala/Typechecker.scala:178:17  0.038
areBothIntOrRealOrChar  match exhaustiveness                                         valid   nativez3   src/main/scala/Typechecker.scala:196:17  0.024
interpret               match exhaustiveness                                         valid   nativez3   src/main/scala/Interpreter.scala:39:25   0.238
isValue                 match exhaustiveness                                         valid   nativez3   src/main/scala/Interpreter.scala:16:17   0.029
next                    match exhaustiveness                                         valid   nativez3   src/main/scala/Interpreter.scala:56:17   0.046
next                    match exhaustiveness                                         valid   nativez3   src/main/scala/Interpreter.scala:70:66   0.049
next                    match exhaustiveness                                         valid   nativez3   src/main/scala/Interpreter.scala:71:73   0.049
next                    match exhaustiveness                                         valid   nativez3   src/main/scala/Interpreter.scala:72:69   0.039
progress                postcondition                                                valid   nativez3   src/main/scala/Progress.scala:26:9       0.941
progress                match exhaustiveness                                         valid   nativez3   src/main/scala/Progress.scala:28:17      0.253
progress                match exhaustiveness                                         valid   nativez3   src/main/scala/Progress.scala:49:33      0.057
progress                match exhaustiveness                                         valid   nativez3   src/main/scala/Progress.scala:50:81      0.044
progress                match exhaustiveness                                         valid   nativez3   src/main/scala/Progress.scala:51:77      0.066
progress                precond. (call check(progress(arg2, t2)))                    valid   nativez3   src/main/scala/Progress.scala:53:67      0.771
progress                precond. (call progress(arg2, t2))                           valid   nativez3   src/main/scala/Progress.scala:53:73      1.728
progress                precond. (call check(progress(arg1, t1)))                    valid   nativez3   src/main/scala/Progress.scala:55:59      0.691
progress                precond. (call progress(arg1, t1))                           valid   nativez3   src/main/scala/Progress.scala:55:65      1.836
progress                precond. (call check(res))                                   valid   nativez3   src/main/scala/Progress.scala:60:49      1.094
progress                precond. (call progress(callee, FunctionType2(get[Type] ...) valid   nativez3   src/main/scala/Progress.scala:61:57      1.169
progress                precond. (call get[Type](tArg1))                             valid   nativez3   src/main/scala/Progress.scala:61:88      0.621
progress                precond. (call get[Type](tArg2))                             valid   nativez3   src/main/scala/Progress.scala:61:99      0.686
subst                   match exhaustiveness                                         valid   nativez3   src/main/scala/Interpreter.scala:352:17  0.053
typecheck               match exhaustiveness                                         valid   nativez3   src/main/scala/Typechecker.scala:20:17   0.035
typecheck               match exhaustiveness                                         valid   nativez3   src/main/scala/Typechecker.scala:65:33   0.066
-------------------------------------------------------------------------------------------------------------------------------------------------------
total: 26   valid: 26   (0 from cache) invalid: 0    unknown: 0    time:  10.801
```

Figure 2: Verification of progress on Lambda2 and Application2

```
stainless summary

areBothBoolean          match exhaustiveness                                valid  nativez3  src/main/scala/Typechecker.scala:145:17   0.214
areBothInt              match exhaustiveness                                valid  nativez3  src/main/scala/Typechecker.scala:161:17   0.014
areBothIntOrReal        match exhaustiveness                                valid  nativez3  src/main/scala/Typechecker.scala:178:17   0.015
areBothIntOrRealOrChar  match exhaustiveness                                valid  nativez3  src/main/scala/Typechecker.scala:196:17   0.018
interpret               match exhaustiveness                                valid  nativez3  src/main/scala/Interpreter.scala:39:25    0.059
isValue                 match exhaustiveness                                valid  nativez3  src/main/scala/Interpreter.scala:16:17    0.016
next                    match exhaustiveness                                valid  nativez3  src/main/scala/Interpreter.scala:56:17    0.058
next                    match exhaustiveness                                valid  nativez3  src/main/scala/Interpreter.scala:85:59    0.069
next                    match exhaustiveness                                valid  nativez3  src/main/scala/Interpreter.scala:93:48    0.088
next                    match exhaustiveness                                valid  nativez3  src/main/scala/Interpreter.scala:94:60    0.081
next                    match exhaustiveness                                valid  nativez3  src/main/scala/Interpreter.scala:99:67    0.042
next                    match exhaustiveness                                valid  nativez3  src/main/scala/Interpreter.scala:107:49   0.097
next                    match exhaustiveness                                valid  nativez3  src/main/scala/Interpreter.scala:108:60   0.075
next                    match exhaustiveness                                valid  nativez3  src/main/scala/Interpreter.scala:113:67   0.087
next                    match exhaustiveness                                valid  nativez3  src/main/scala/Interpreter.scala:121:43   0.021
next                    match exhaustiveness                                valid  nativez3  src/main/scala/Interpreter.scala:127:49   0.018
next                    match exhaustiveness                                valid  nativez3  src/main/scala/Interpreter.scala:128:60   0.018
next                    match exhaustiveness                                valid  nativez3  src/main/scala/Interpreter.scala:133:67   0.020
next                    match exhaustiveness                                valid  nativez3  src/main/scala/Interpreter.scala:141:52   0.015
next                    match exhaustiveness                                valid  nativez3  src/main/scala/Interpreter.scala:142:60   0.016
next                    division by zero                                    valid  nativez3  src/main/scala/Interpreter.scala:144:88   0.216
next                    match exhaustiveness                                valid  nativez3  src/main/scala/Interpreter.scala:148:67   0.019
next                    match exhaustiveness                                valid  nativez3  src/main/scala/Interpreter.scala:157:53   0.021
next                    match exhaustiveness                                valid  nativez3  src/main/scala/Interpreter.scala:158:60   0.018
next                    remainder by zero                                   valid  nativez3  src/main/scala/Interpreter.scala:160:88   0.217
next                    match exhaustiveness                                valid  nativez3  src/main/scala/Interpreter.scala:167:50   0.033
next                    match exhaustiveness                                valid  nativez3  src/main/scala/Interpreter.scala:168:60   0.029
next                    modulo by zero                                      valid  nativez3  src/main/scala/Interpreter.scala:170:107  0.189
next                    modulo by zero                                      valid  nativez3  src/main/scala/Interpreter.scala:171:88   0.196
next                    match exhaustiveness                                valid  nativez3  src/main/scala/Interpreter.scala:180:56   0.030
next                    match exhaustiveness                                valid  nativez3  src/main/scala/Interpreter.scala:182:41   0.023
next                    match exhaustiveness                                valid  nativez3  src/main/scala/Interpreter.scala:190:58   0.019
next                    match exhaustiveness                                valid  nativez3  src/main/scala/Interpreter.scala:192:41   0.015
next                    match exhaustiveness                                valid  nativez3  src/main/scala/Interpreter.scala:194:57   0.025
next                    match exhaustiveness                                valid  nativez3  src/main/scala/Interpreter.scala:206:49   0.015
progress                postcondition                                       valid  nativez3  src/main/scala/Progress.scala:26:9        9885.492
progress                match exhaustiveness                                valid  nativez3  src/main/scala/Progress.scala:28:17       120.678
progress                precond. (call check(res))                          valid  nativez3  src/main/scala/Progress.scala:67:33       135.991
progress                match exhaustiveness                                valid  nativez3  src/main/scala/Progress.scala:70:33       0.037
progress                precond. (call check(progress(thenn, t)))           valid  nativez3  src/main/scala/Progress.scala:72:78       1508.200
progress                precond. (call progress(thenn, t))                  valid  nativez3  src/main/scala/Progress.scala:72:84       136.448
progress                precond. (call check(progress(elze, t)))            valid  nativez3  src/main/scala/Progress.scala:74:79       1499.534
progress                precond. (call progress(elze, t))                   valid  nativez3  src/main/scala/Progress.scala:74:85       136.516
progress                precond. (call check(progress(cond, BooleanType())))  valid  nativez3  src/main/scala/Progress.scala:75:59     1512.209
progress                precond. (call progress(cond, BooleanType()))       valid  nativez3  src/main/scala/Progress.scala:75:65       135.164
progress                precond. (call check(res))                          valid  nativez3  src/main/scala/Progress.scala:80:33       139.125
progress                match exhaustiveness                                valid  nativez3  src/main/scala/Progress.scala:83:33       0.033
progress                precond. (call check(progress(rhs, t)))             valid  nativez3  src/main/scala/Progress.scala:86:72       1490.709
progress                precond. (call progress(rhs, t))                    valid  nativez3  src/main/scala/Progress.scala:86:78       129.967
progress                precond. (call check(progress(rhs, t)))             valid  nativez3  src/main/scala/Progress.scala:87:73       1501.996
progress                precond. (call progress(rhs, t))                    valid  nativez3  src/main/scala/Progress.scala:87:79       129.705
progress                precond. (call check(progress(lhs, t)))             valid  nativez3  src/main/scala/Progress.scala:88:56       1505.890
progress                precond. (call progress(lhs, t))                    valid  nativez3  src/main/scala/Progress.scala:88:62       128.896
progress                precond. (call check(res))                          valid  nativez3  src/main/scala/Progress.scala:92:33       132.570
progress                match exhaustiveness                                valid  nativez3  src/main/scala/Progress.scala:95:33       0.037
progress                precond. (call check(progress(rhs, t)))             valid  nativez3  src/main/scala/Progress.scala:98:72       1649.960
progress                precond. (call progress(rhs, t))                    valid  nativez3  src/main/scala/Progress.scala:98:78       129.119
progress                precond. (call check(progress(rhs, t)))             valid  nativez3  src/main/scala/Progress.scala:99:73       1652.393
progress                precond. (call progress(rhs, t))                    valid  nativez3  src/main/scala/Progress.scala:99:79       128.216
progress                precond. (call check(progress(lhs, t)))             valid  nativez3  src/main/scala/Progress.scala:100:56      1641.018
progress                precond. (call progress(lhs, t))                    valid  nativez3  src/main/scala/Progress.scala:100:62      127.853
progress                precond. (call check(res))                          valid  nativez3  src/main/scala/Progress.scala:104:33      130.821
progress                match exhaustiveness                                valid  nativez3  src/main/scala/Progress.scala:106:33      0.031
progress                precond. (call check(progress(e, t)))               valid  nativez3  src/main/scala/Progress.scala:109:53      1641.897
progress                precond. (call progress(e, t))                      valid  nativez3  src/main/scala/Progress.scala:109:59      129.649
progress                precond. (call check(res))                          valid  nativez3  src/main/scala/Progress.scala:113:33      131.454
progress                match exhaustiveness                                valid  nativez3  src/main/scala/Progress.scala:116:33      0.038
progress                precond. (call check(progress(rhs, t)))             valid  nativez3  src/main/scala/Progress.scala:119:72      1652.756
progress                precond. (call progress(rhs, t))                    valid  nativez3  src/main/scala/Progress.scala:119:78      128.983
progress                precond. (call check(progress(rhs, t)))             valid  nativez3  src/main/scala/Progress.scala:120:73      1654.182
progress                precond. (call progress(rhs, t))                    valid  nativez3  src/main/scala/Progress.scala:120:79      128.187
progress                precond. (call check(progress(lhs, t)))             valid  nativez3  src/main/scala/Progress.scala:121:56      1640.785
progress                precond. (call progress(lhs, t))                    valid  nativez3  src/main/scala/Progress.scala:121:62      128.137
progress                precond. (call check(res))                          valid  nativez3  src/main/scala/Progress.scala:125:33      130.193
progress                match exhaustiveness                                valid  nativez3  src/main/scala/Progress.scala:128:33      0.035
progress                precond. (call check(progress(rhs, t)))             valid  nativez3  src/main/scala/Progress.scala:131:72      1638.195
progress                precond. (call progress(rhs, t))                    valid  nativez3  src/main/scala/Progress.scala:131:78      128.205
progress                precond. (call check(progress(rhs, t)))             valid  nativez3  src/main/scala/Progress.scala:132:73      1649.007
progress                precond. (call progress(rhs, t))                    valid  nativez3  src/main/scala/Progress.scala:132:79      129.011
progress                precond. (call check(progress(lhs, t)))             valid  nativez3  src/main/scala/Progress.scala:133:56      1639.504
progress                precond. (call progress(lhs, t))                    valid  nativez3  src/main/scala/Progress.scala:133:62      129.182
progress                precond. (call check(res))                          valid  nativez3  src/main/scala/Progress.scala:137:33      131.356
progress                match exhaustiveness                                valid  nativez3  src/main/scala/Progress.scala:140:33      0.034
progress                precond. (call check(progress(rhs, t)))             valid  nativez3  src/main/scala/Progress.scala:142:72      1653.870
progress                precond. (call progress(rhs, t))                    valid  nativez3  src/main/scala/Progress.scala:142:78      129.254
progress                precond. (call check(progress(lhs, t)))             valid  nativez3  src/main/scala/Progress.scala:143:56      1640.766
progress                precond. (call progress(lhs, t))                    valid  nativez3  src/main/scala/Progress.scala:143:62      129.671
progress                precond. (call check(res))                          valid  nativez3  src/main/scala/Progress.scala:147:33      132.224
progress                match exhaustiveness                                valid  nativez3  src/main/scala/Progress.scala:150:33      0.036
progress                precond. (call check(progress(rhs, t)))             valid  nativez3  src/main/scala/Progress.scala:152:72      1640.855
progress                precond. (call progress(rhs, t))                    valid  nativez3  src/main/scala/Progress.scala:152:78      128.834
progress                precond. (call check(progress(lhs, t)))             valid  nativez3  src/main/scala/Progress.scala:153:56      1648.842
progress                precond. (call progress(lhs, t))                    valid  nativez3  src/main/scala/Progress.scala:153:62      129.052
progress                precond. (call check(res))                          valid  nativez3  src/main/scala/Progress.scala:159:33      126.900
progress                match exhaustiveness                                valid  nativez3  src/main/scala/Progress.scala:162:33      0.038
progress                precond. (call check(progress(rhs, t)))             valid  nativez3  src/main/scala/Progress.scala:164:71      1655.203
progress                precond. (call progress(rhs, t))                    valid  nativez3  src/main/scala/Progress.scala:164:77      127.417
progress                precond. (call check(progress(lhs, t)))             valid  nativez3  src/main/scala/Progress.scala:165:56      1610.654
progress                precond. (call progress(lhs, t))                    valid  nativez3  src/main/scala/Progress.scala:165:62      128.015
progress                precond. (call check(res))                          valid  nativez3  src/main/scala/Progress.scala:169:33      127.397
progress                match exhaustiveness                                valid  nativez3  src/main/scala/Progress.scala:173:33      0.038
progress                precond. (call check(progress(end, IntegerType())))    valid  nativez3  src/main/scala/Progress.scala:176:49   1446.098
progress                precond. (call progress(end, IntegerType()))        valid  nativez3  src/main/scala/Progress.scala:176:55      127.868
progress                precond. (call check(progress(start, IntegerType())))  valid  nativez3  src/main/scala/Progress.scala:178:49   1403.438
progress                precond. (call progress(start, IntegerType()))      valid  nativez3  src/main/scala/Progress.scala:178:55      128.198
progress                precond. (call check(progress(e, t)))               valid  nativez3  src/main/scala/Progress.scala:179:59      1360.192
progress                precond. (call progress(e, t))                      valid  nativez3  src/main/scala/Progress.scala:179:65      127.965
progress                precond. (call check(res))                          valid  nativez3  src/main/scala/Progress.scala:183:33      127.507
progress                match exhaustiveness                                valid  nativez3  src/main/scala/Progress.scala:185:33      0.034
progress                precond. (call check(progress(e, StringType())))    valid  nativez3  src/main/scala/Progress.scala:187:53      1298.050
progress                precond. (call progress(e, StringType()))           valid  nativez3  src/main/scala/Progress.scala:187:59      128.050
subst                   match exhaustiveness                                valid  nativez3  src/main/scala/Interpreter.scala:352:17   0.021
typecheck               match exhaustiveness                                valid  nativez3  src/main/scala/Typechecker.scala:20:17    0.021
typecheck               match exhaustiveness                                valid  nativez3  src/main/scala/Typechecker.scala:40:45    0.012
typecheck               match exhaustiveness                                valid  nativez3  src/main/scala/Typechecker.scala:41:41    0.015
---------------------------------------------------------------------------------------------------------------------------------------------
total: 115  valid: 115  (0 from cache) invalid: 0    unknown: 0    time: 55658.039
```

Figure 3: Verification of progress on if-then-else expressions, general arithmetic and string operations

```
stainless summary

areBothBoolean          match exhaustiveness                                  valid  nativez3  src/main/scala/Typechecker.scala:145:17  0.015
areBothInt              match exhaustiveness                                  valid  nativez3  src/main/scala/Typechecker.scala:161:17  0.014
areBothIntOrReal        match exhaustiveness                                  valid  nativez3  src/main/scala/Typechecker.scala:178:17  0.014
areBothIntOrRealOrChar  match exhaustiveness                                  valid  nativez3  src/main/scala/Typechecker.scala:196:17  0.014
interpret               match exhaustiveness                                  valid  nativez3  src/main/scala/Interpreter.scala:39:25   0.621
isValue                 match exhaustiveness                                  valid  nativez3  src/main/scala/Interpreter.scala:16:17   0.019
next                    match exhaustiveness                                  valid  nativez3  src/main/scala/Interpreter.scala:56:17   0.176
next                    match exhaustiveness                                  valid  nativez3  src/main/scala/Interpreter.scala:218:52  0.141
next                    match exhaustiveness                                  valid  nativez3  src/main/scala/Interpreter.scala:219:60  0.136
next                    match exhaustiveness                                  valid  nativez3  src/main/scala/Interpreter.scala:224:67  0.174
next                    match exhaustiveness                                  valid  nativez3  src/main/scala/Interpreter.scala:229:57  0.139
next                    match exhaustiveness                                  valid  nativez3  src/main/scala/Interpreter.scala:237:55  0.155
next                    match exhaustiveness                                  valid  nativez3  src/main/scala/Interpreter.scala:238:60  0.143
next                    match exhaustiveness                                  valid  nativez3  src/main/scala/Interpreter.scala:243:67  0.018
next                    match exhaustiveness                                  valid  nativez3  src/main/scala/Interpreter.scala:248:57  0.022
next                    match exhaustiveness                                  valid  nativez3  src/main/scala/Interpreter.scala:256:54  0.022
next                    match exhaustiveness                                  valid  nativez3  src/main/scala/Interpreter.scala:257:60  0.017
next                    match exhaustiveness                                  valid  nativez3  src/main/scala/Interpreter.scala:262:67  0.018
next                    match exhaustiveness                                  valid  nativez3  src/main/scala/Interpreter.scala:267:57  0.020
next                    match exhaustiveness                                  valid  nativez3  src/main/scala/Interpreter.scala:275:57  0.022
next                    match exhaustiveness                                  valid  nativez3  src/main/scala/Interpreter.scala:276:60  0.017
next                    match exhaustiveness                                  valid  nativez3  src/main/scala/Interpreter.scala:281:67  0.015
next                    match exhaustiveness                                  valid  nativez3  src/main/scala/Interpreter.scala:286:57  0.018
next                    match exhaustiveness                                  valid  nativez3  src/main/scala/Interpreter.scala:297:47  0.015
next                    match exhaustiveness                                  valid  nativez3  src/main/scala/Interpreter.scala:300:41  0.020
next                    match exhaustiveness                                  valid  nativez3  src/main/scala/Interpreter.scala:309:46  0.026
next                    match exhaustiveness                                  valid  nativez3  src/main/scala/Interpreter.scala:312:41  0.021
next                    match exhaustiveness                                  valid  nativez3  src/main/scala/Interpreter.scala:321:51  0.022
next                    match exhaustiveness                                  valid  nativez3  src/main/scala/Interpreter.scala:324:41  0.022
next                    match exhaustiveness                                  valid  nativez3  src/main/scala/Interpreter.scala:333:40  0.022
progress                postcondition                                         valid  nativez3  src/main/scala/Progress.scala:26:9       12281.193
progress                match exhaustiveness                                  valid  nativez3  src/main/scala/Progress.scala:28:17      36.494
progress                match exhaustiveness                                  valid  nativez3  src/main/scala/Progress.scala:31:33      0.013
progress                precond. (call check(progress(value, tValue)))        valid  nativez3  src/main/scala/Progress.scala:33:51      4445.781
progress                precond. (call progress(value, tValue))               valid  nativez3  src/main/scala/Progress.scala:33:57      38.655
progress                precond. (call check(res))                            valid  nativez3  src/main/scala/Progress.scala:195:33     40.134
progress                match exhaustiveness                                  valid  nativez3  src/main/scala/Progress.scala:196:33     0.030
progress                precond. (call check(progress(rhs, get[Type](t2))))   valid  nativez3  src/main/scala/Progress.scala:198:73     4495.762
progress                precond. (call progress(rhs, get[Type](t2)))          valid  nativez3  src/main/scala/Progress.scala:198:79     38.605
progress                precond. (call get[Type](t2))                         valid  nativez3  src/main/scala/Progress.scala:198:93     38.827
progress                precond. (call check(progress(lhs, get[Type](t1))))   valid  nativez3  src/main/scala/Progress.scala:199:56     4457.633
progress                precond. (call progress(lhs, get[Type](t1)))          valid  nativez3  src/main/scala/Progress.scala:199:62     38.927
progress                precond. (call get[Type](t1))                         valid  nativez3  src/main/scala/Progress.scala:199:76     36.384
progress                precond. (call check(res))                            valid  nativez3  src/main/scala/Progress.scala:205:33     37.889
progress                match exhaustiveness                                  valid  nativez3  src/main/scala/Progress.scala:209:33     0.033
progress                precond. (call check(progress(rhs, get[Type](t2))))   valid  nativez3  src/main/scala/Progress.scala:213:72     4493.361
progress                precond. (call progress(rhs, get[Type](t2)))          valid  nativez3  src/main/scala/Progress.scala:213:78     36.930
progress                precond. (call get[Type](t2))                         valid  nativez3  src/main/scala/Progress.scala:213:92     37.722
progress                precond. (call check(progress(rhs, get[Type](t2))))   valid  nativez3  src/main/scala/Progress.scala:214:73     4460.533
progress                precond. (call progress(rhs, get[Type](t2)))          valid  nativez3  src/main/scala/Progress.scala:214:79     36.715
progress                precond. (call get[Type](t2))                         valid  nativez3  src/main/scala/Progress.scala:214:93     37.534
progress                precond. (call check(progress(rhs, get[Type](t2))))   valid  nativez3  src/main/scala/Progress.scala:215:69     4502.922
progress                precond. (call progress(rhs, get[Type](t2)))          valid  nativez3  src/main/scala/Progress.scala:215:75     36.989
progress                precond. (call get[Type](t2))                         valid  nativez3  src/main/scala/Progress.scala:215:89     37.563
progress                precond. (call check(progress(lhs, get[Type](t1))))   valid  nativez3  src/main/scala/Progress.scala:216:56     4505.964
progress                precond. (call progress(lhs, get[Type](t1)))          valid  nativez3  src/main/scala/Progress.scala:216:62     36.218
progress                precond. (call get[Type](t1))                         valid  nativez3  src/main/scala/Progress.scala:216:76     37.208
progress                precond. (call check(res))                            valid  nativez3  src/main/scala/Progress.scala:222:33     37.078
progress                match exhaustiveness                                  valid  nativez3  src/main/scala/Progress.scala:226:33     0.034
progress                precond. (call check(progress(rhs, get[Type](t2))))   valid  nativez3  src/main/scala/Progress.scala:230:72     4449.978
progress                precond. (call progress(rhs, get[Type](t2)))          valid  nativez3  src/main/scala/Progress.scala:230:78     36.012
progress                precond. (call get[Type](t2))                         valid  nativez3  src/main/scala/Progress.scala:230:92     37.181
progress                precond. (call check(progress(rhs, get[Type](t2))))   valid  nativez3  src/main/scala/Progress.scala:231:73     4463.734
progress                precond. (call progress(rhs, get[Type](t2)))          valid  nativez3  src/main/scala/Progress.scala:231:79     36.016
progress                precond. (call get[Type](t2))                         valid  nativez3  src/main/scala/Progress.scala:231:93     37.208
progress                precond. (call check(progress(rhs, get[Type](t2))))   valid  nativez3  src/main/scala/Progress.scala:232:69     4476.135
progress                precond. (call progress(rhs, get[Type](t2)))          valid  nativez3  src/main/scala/Progress.scala:232:75     36.305
progress                precond. (call get[Type](t2))                         valid  nativez3  src/main/scala/Progress.scala:232:89     37.238
progress                precond. (call check(progress(lhs, get[Type](t1))))   valid  nativez3  src/main/scala/Progress.scala:233:56     4493.750
progress                precond. (call progress(lhs, get[Type](t1)))          valid  nativez3  src/main/scala/Progress.scala:233:62     36.634
progress                precond. (call get[Type](t1))                         valid  nativez3  src/main/scala/Progress.scala:233:76     37.745
progress                precond. (call check(res))                            valid  nativez3  src/main/scala/Progress.scala:239:33     38.019
progress                match exhaustiveness                                  valid  nativez3  src/main/scala/Progress.scala:243:33     0.037
progress                precond. (call check(progress(rhs, get[Type](t2))))   valid  nativez3  src/main/scala/Progress.scala:247:72     4523.903
progress                precond. (call progress(rhs, get[Type](t2)))          valid  nativez3  src/main/scala/Progress.scala:247:78     36.346
progress                precond. (call get[Type](t2))                         valid  nativez3  src/main/scala/Progress.scala:247:92     37.627
progress                precond. (call check(progress(rhs, get[Type](t2))))   valid  nativez3  src/main/scala/Progress.scala:248:73     4512.741
progress                precond. (call progress(rhs, get[Type](t2)))          valid  nativez3  src/main/scala/Progress.scala:248:79     36.465
progress                precond. (call get[Type](t2))                         valid  nativez3  src/main/scala/Progress.scala:248:93     37.228
progress                precond. (call check(progress(rhs, get[Type](t2))))   valid  nativez3  src/main/scala/Progress.scala:249:69     4540.012
progress                precond. (call progress(rhs, get[Type](t2)))          valid  nativez3  src/main/scala/Progress.scala:249:75     36.847
progress                precond. (call get[Type](t2))                         valid  nativez3  src/main/scala/Progress.scala:249:89     37.193
progress                precond. (call check(progress(lhs, get[Type](t1))))   valid  nativez3  src/main/scala/Progress.scala:250:56     4533.282
progress                precond. (call progress(lhs, get[Type](t1)))          valid  nativez3  src/main/scala/Progress.scala:250:62     36.089
progress                precond. (call get[Type](t1))                         valid  nativez3  src/main/scala/Progress.scala:250:76     37.120
progress                precond. (call check(res))                            valid  nativez3  src/main/scala/Progress.scala:256:33     37.286
progress                match exhaustiveness                                  valid  nativez3  src/main/scala/Progress.scala:260:33     0.033
progress                precond. (call check(progress(rhs, get[Type](t2))))   valid  nativez3  src/main/scala/Progress.scala:264:72     4484.786
progress                precond. (call progress(rhs, get[Type](t2)))          valid  nativez3  src/main/scala/Progress.scala:264:78     35.760
progress                precond. (call get[Type](t2))                         valid  nativez3  src/main/scala/Progress.scala:264:92     36.688
progress                precond. (call check(progress(rhs, get[Type](t2))))   valid  nativez3  src/main/scala/Progress.scala:265:73     4476.096
progress                precond. (call progress(rhs, get[Type](t2)))          valid  nativez3  src/main/scala/Progress.scala:265:79     35.695
progress                precond. (call get[Type](t2))                         valid  nativez3  src/main/scala/Progress.scala:265:93     37.000
progress                precond. (call check(progress(rhs, get[Type](t2))))   valid  nativez3  src/main/scala/Progress.scala:266:69     4462.621
progress                precond. (call progress(rhs, get[Type](t2)))          valid  nativez3  src/main/scala/Progress.scala:266:75     35.992
progress                precond. (call get[Type](t2))                         valid  nativez3  src/main/scala/Progress.scala:266:89     37.141
progress                precond. (call check(progress(lhs, get[Type](t1))))   valid  nativez3  src/main/scala/Progress.scala:267:56     4528.307
progress                precond. (call progress(lhs, get[Type](t1)))          valid  nativez3  src/main/scala/Progress.scala:267:62     36.757
progress                precond. (call get[Type](t1))                         valid  nativez3  src/main/scala/Progress.scala:267:76     37.498
progress                precond. (call check(res))                            valid  nativez3  src/main/scala/Progress.scala:273:33     37.597
progress                match exhaustiveness                                  valid  nativez3  src/main/scala/Progress.scala:276:33     0.032
progress                precond. (call check(progress(rhs, t)))               valid  nativez3  src/main/scala/Progress.scala:279:75     4521.057
progress                precond. (call progress(rhs, t))                      valid  nativez3  src/main/scala/Progress.scala:279:81     36.279
progress                precond. (call check(progress(lhs, t)))               valid  nativez3  src/main/scala/Progress.scala:280:56     4545.779
progress                precond. (call progress(lhs, t))                      valid  nativez3  src/main/scala/Progress.scala:280:62     36.237
progress                precond. (call check(res))                            valid  nativez3  src/main/scala/Progress.scala:284:33     37.667
progress                match exhaustiveness                                  valid  nativez3  src/main/scala/Progress.scala:287:33     0.041
progress                precond. (call check(progress(rhs, t)))               valid  nativez3  src/main/scala/Progress.scala:290:76     4493.965
progress                precond. (call progress(rhs, t))                      valid  nativez3  src/main/scala/Progress.scala:290:82     36.621
progress                precond. (call check(progress(lhs, t)))               valid  nativez3  src/main/scala/Progress.scala:291:56     4518.090
progress                precond. (call progress(lhs, t))                      valid  nativez3  src/main/scala/Progress.scala:291:62     36.169
progress                precond. (call check(res))                            valid  nativez3  src/main/scala/Progress.scala:295:33     37.351
progress                match exhaustiveness                                  valid  nativez3  src/main/scala/Progress.scala:298:33     0.032
progress                precond. (call check(progress(rhs, t)))               valid  nativez3  src/main/scala/Progress.scala:301:75     4456.905
progress                precond. (call progress(rhs, t))                      valid  nativez3  src/main/scala/Progress.scala:301:81     36.119
progress                precond. (call check(progress(lhs, t)))               valid  nativez3  src/main/scala/Progress.scala:302:56     3005.746
progress                precond. (call progress(lhs, t))                      valid  nativez3  src/main/scala/Progress.scala:302:62     36.562
progress                precond. (call check(res))                            valid  nativez3  src/main/scala/Progress.scala:306:33     36.953
progress                match exhaustiveness                                  valid  nativez3  src/main/scala/Progress.scala:308:33     0.031
progress                precond. (call check(progress(e, t)))                 valid  nativez3  src/main/scala/Progress.scala:310:53     2975.191
progress                precond. (call progress(e, t))                        valid  nativez3  src/main/scala/Progress.scala:310:59     36.328
subst                   match exhaustiveness                                  valid  nativez3  src/main/scala/Interpreter.scala:352:17  0.018
typecheck               match exhaustiveness                                  valid  nativez3  src/main/scala/Typechecker.scala:20:17   0.021
typecheck               match exhaustiveness                                  valid  nativez3  src/main/scala/Typechecker.scala:106:33  0.013
-----------------------------------------------------------------------------------------------------------------------------------------------
total: 124  valid: 124  (0 from cache) invalid: 0    unknown: 0    time: 128108.538
```

Figure 4: Verification of progress on Let expressions, comparisons and logical operators

```
stainless summary

areBothBoolean          match exhaustiveness            valid  nativez3  src/main/scala/Typechecker.scala:144:17   0.201
areBothInt              match exhaustiveness            valid  nativez3  src/main/scala/Typechecker.scala:160:17   0.154
areBothIntOrReal        match exhaustiveness            valid  nativez3  src/main/scala/Typechecker.scala:177:17   0.253
areBothIntOrRealOrChar  match exhaustiveness            valid  nativez3  src/main/scala/Typechecker.scala:195:17   0.028
helperArgs              match exhaustiveness            valid  nativez3  src/main/scala/Interpreter.scala:72:49    0.018
helperArgs              match exhaustiveness            valid  nativez3  src/main/scala/Interpreter.scala:78:80    0.189
helperArgs              match exhaustiveness            valid  nativez3  src/main/scala/Interpreter.scala:82:83    0.152
helperBody              match exhaustiveness            valid  nativez3  src/main/scala/Interpreter.scala:90:49    0.016
helperNewEnv            match exhaustiveness            valid  nativez3  src/main/scala/Typechecker.scala:53:49    0.023
interpret               match exhaustiveness            valid  nativez3  src/main/scala/Interpreter.scala:40:25    0.185
isValue                 match exhaustiveness            valid  nativez3  src/main/scala/Interpreter.scala:18:17    0.034
next                    match exhaustiveness            valid  nativez3  src/main/scala/Interpreter.scala:57:17    0.088
next                    match exhaustiveness            valid  nativez3  src/main/scala/Interpreter.scala:63:59    0.428
next                    match exhaustiveness            valid  nativez3  src/main/scala/Interpreter.scala:97:41    0.409
next                    match exhaustiveness            valid  nativez3  src/main/scala/Interpreter.scala:113:59   0.570
next                    match exhaustiveness            valid  nativez3  src/main/scala/Interpreter.scala:121:48   0.259
next                    match exhaustiveness            valid  nativez3  src/main/scala/Interpreter.scala:122:60   0.333
next                    match exhaustiveness            valid  nativez3  src/main/scala/Interpreter.scala:127:67   0.102
next                    match exhaustiveness            valid  nativez3  src/main/scala/Interpreter.scala:135:49   0.258
next                    match exhaustiveness            valid  nativez3  src/main/scala/Interpreter.scala:136:60   0.166
next                    match exhaustiveness            valid  nativez3  src/main/scala/Interpreter.scala:141:67   0.047
next                    match exhaustiveness            valid  nativez3  src/main/scala/Interpreter.scala:149:43   0.046
next                    match exhaustiveness            valid  nativez3  src/main/scala/Interpreter.scala:155:49   0.052
next                    match exhaustiveness            valid  nativez3  src/main/scala/Interpreter.scala:156:60   0.050
next                    match exhaustiveness            valid  nativez3  src/main/scala/Interpreter.scala:161:67   0.064
next                    match exhaustiveness            valid  nativez3  src/main/scala/Interpreter.scala:169:52   0.074
next                    match exhaustiveness            valid  nativez3  src/main/scala/Interpreter.scala:170:60   0.041
next                    division by zero                valid  nativez3  src/main/scala/Interpreter.scala:172:88   0.430
next                    match exhaustiveness            valid  nativez3  src/main/scala/Interpreter.scala:176:67   0.053
next                    match exhaustiveness            valid  nativez3  src/main/scala/Interpreter.scala:185:53   0.072
next                    match exhaustiveness            valid  nativez3  src/main/scala/Interpreter.scala:186:60   0.039
next                    remainder by zero               valid  nativez3  src/main/scala/Interpreter.scala:188:88   0.335
next                    match exhaustiveness            valid  nativez3  src/main/scala/Interpreter.scala:195:50   0.046
next                    match exhaustiveness            valid  nativez3  src/main/scala/Interpreter.scala:196:60   0.041
next                    modulo by zero                  valid  nativez3  src/main/scala/Interpreter.scala:198:107  0.166
next                    modulo by zero                  valid  nativez3  src/main/scala/Interpreter.scala:199:88   0.064
next                    match exhaustiveness            valid  nativez3  src/main/scala/Interpreter.scala:208:56   0.038
next                    match exhaustiveness            valid  nativez3  src/main/scala/Interpreter.scala:210:41   0.046
next                    match exhaustiveness            valid  nativez3  src/main/scala/Interpreter.scala:218:58   0.041
next                    match exhaustiveness            valid  nativez3  src/main/scala/Interpreter.scala:220:41   0.039
next                    match exhaustiveness            valid  nativez3  src/main/scala/Interpreter.scala:222:57   0.049
next                    match exhaustiveness            valid  nativez3  src/main/scala/Interpreter.scala:234:49   0.051
next                    match exhaustiveness            valid  nativez3  src/main/scala/Interpreter.scala:246:52   0.046
next                    match exhaustiveness            valid  nativez3  src/main/scala/Interpreter.scala:247:60   0.057
next                    match exhaustiveness            valid  nativez3  src/main/scala/Interpreter.scala:252:67   0.035
next                    match exhaustiveness            valid  nativez3  src/main/scala/Interpreter.scala:257:57   0.039
next                    match exhaustiveness            valid  nativez3  src/main/scala/Interpreter.scala:265:55   0.040
next                    match exhaustiveness            valid  nativez3  src/main/scala/Interpreter.scala:266:60   0.070
next                    match exhaustiveness            valid  nativez3  src/main/scala/Interpreter.scala:271:67   0.083
next                    match exhaustiveness            valid  nativez3  src/main/scala/Interpreter.scala:276:57   0.038
next                    match exhaustiveness            valid  nativez3  src/main/scala/Interpreter.scala:284:54   0.050
next                    match exhaustiveness            valid  nativez3  src/main/scala/Interpreter.scala:285:60   0.031
next                    match exhaustiveness            valid  nativez3  src/main/scala/Interpreter.scala:290:67   0.042
next                    match exhaustiveness            valid  nativez3  src/main/scala/Interpreter.scala:295:57   0.031
next                    match exhaustiveness            valid  nativez3  src/main/scala/Interpreter.scala:303:57   0.030
next                    match exhaustiveness            valid  nativez3  src/main/scala/Interpreter.scala:304:60   0.040
next                    match exhaustiveness            valid  nativez3  src/main/scala/Interpreter.scala:309:67   0.044
next                    match exhaustiveness            valid  nativez3  src/main/scala/Interpreter.scala:314:57   0.042
next                    match exhaustiveness            valid  nativez3  src/main/scala/Interpreter.scala:325:47   0.044
next                    match exhaustiveness            valid  nativez3  src/main/scala/Interpreter.scala:328:41   0.044
next                    match exhaustiveness            valid  nativez3  src/main/scala/Interpreter.scala:337:46   0.036
next                    match exhaustiveness            valid  nativez3  src/main/scala/Interpreter.scala:340:41   0.036
next                    match exhaustiveness            valid  nativez3  src/main/scala/Interpreter.scala:349:51   0.035
next                    match exhaustiveness            valid  nativez3  src/main/scala/Interpreter.scala:352:41   0.040
next                    match exhaustiveness            valid  nativez3  src/main/scala/Interpreter.scala:361:40   0.042
subst                   match exhaustiveness            valid  nativez3  src/main/scala/Interpreter.scala:380:17   0.030
typecheck               match exhaustiveness            valid  nativez3  src/main/scala/Typechecker.scala:21:17    0.015
typecheck               match exhaustiveness            valid  nativez3  src/main/scala/Typechecker.scala:41:45    0.078
typecheck               match exhaustiveness            valid  nativez3  src/main/scala/Typechecker.scala:42:41    0.255
typecheck               match exhaustiveness            valid  nativez3  src/main/scala/Typechecker.scala:63:33    0.109
typecheck               match exhaustiveness            valid  nativez3  src/main/scala/Typechecker.scala:67:64    0.089
typecheck               precond. (call get[Type](aType))  valid  nativez3  src/main/scala/Typechecker.scala:67:102  2616.159
typecheck               match exhaustiveness            valid  nativez3  src/main/scala/Typechecker.scala:105:33   0.088
------------------------------------------------------------------------------------------------------------------------
total: 73   valid: 73   (0 from cache) invalid: 0    unknown: 0    time: 2623.568
```

Figure 5: Verification of the files Interpreter, Typechecker, Types, Expressions and Identifiers with all the expressions