Who checks the typecheckers?

Bounded verification of type systems with symbolic execution

Kartik Chandra, Henry M. Gunn High School, Palo Alto
Rastislav Bodik, University of Washington, Seattle

PNW Programming Languages and Software Engineering Meeting
March 15, 2016
Today, type systems are not checked

Most common programming languages such as Java, Scala, etc. do not have verified type systems.

Bugs in the typechecker could allow incorrect programs to be executed.

program checking hierarchy

Language designer

A human writes the type checker

Typechecker

The type checker verifies the program

Program

The program controls sensitive operations

High assurance system
High-priority type safety issues (our long-term goal)

Since Rust’s typechecker has not been formally verified, there are many bugs. Here is a small selection of recently-reported high-priority type bugs in Rust, many of which have been fixed.
High-priority type safety issues (our long-term goal)

So, as an unexpected result of a fun conversation last week, Nada and I managed to prove Scala’s type system unsound:

```scala
trait A { type L <: Nothing }
trait B { type L ::= Any

def toL(b: B)(x: Any) : b.L = x
val p: B with A = null

// we can create a value of type Nothing
println(toL(p)("hello") : Nothing)

// at runtime: java.lang.ClassCastException: java.lang.String cannot be cast to scala.runtime.Nothing$
```
Motivating Example

class CHEF
feature
  cook(x: INGREDIENT): FOOD is ...
End

class BAKER
feature
  cook(x: DOUGH): BREAD is
    DO knead(x); ...; END
End

b : BAKER
c : CHEF
-- Eiffel allows this!
c := b
c.cook(cauliflower)
-- ERROR: cannot knead cauliflower

- Eiffel allows covariant method arguments!
- This is unsafe and leads to incorrect programs passing the typechecker.
- This is hard to fix because of legacy code breaking.
Existing model checking of type systems

For all intermediate states $\Sigma$ that meet the invariant… For all possible steps…

(Explicit invariant, based on Korat checking)

[Roberson et al, OOPSLA'08]
Existing model checking of type systems, vs. ours

For all intermediate states $\Sigma$ that meet the invariant…
For all possible steps…

(Explicit invariant, based on Korat checking)

Abstract state $\Sigma$ \hspace{1cm} Abstract state $\Sigma^*$
Invariant
Intermediate concrete state $\Sigma$

Does the invariant hold?

Dynamic types $\Sigma^*$

[Roberson et al, OOPSLA'08]

For all programs $P$ that type check, does executing $P$ go wrong?

(Without invariant, just run on the Rosette checker)

Initial concrete state $P$

Step*

... Program output $P^*$

Was there a run-time error?
Our setup: symbolic typechecker checking for free

Rosette, a symbolic execution system for Racket, lets us run the typechecker-interpreter chain symbolically with very few changes.
Specifying the type system is easy

(From Types and Programming Languages by Benjamin Pierce.)

t ::= true false if t then t else t 0 succ t pred t iszero t

(define arithmetic-stx '(((t ->
  | true  | false
  | (if t t t)
  | zero
  | (succ t)
  | (pred t)
  | (iszero? t))))

(define (type-if pred left right) (if (and
  (eq? (type-expr pred) 'bool)
  (eq? (type-expr left) (type-expr right)))
  (type-expr left)
  (assert #f)))

...
Bad typechecker

--snip--

(define (type-if pred left right)
  (if
   (eq? (type-expr pred) 'bool)
   (begin
    (type-expr left)
    (type-expr right))
   (assert #f)))

--snip--
Bad typechecker

--snip--

(define (type-if pred left right)
  (if
   (eq? (type-expr pred) 'bool)
   (begin
    (type-expr left)
    (type-expr right))
   (assert #f)))

--snip--

$ racket arithmetic.rkt
verify: counterexample found:

'(iszero?
  (if
   (iszero? (succ zero))
   (if (iszero? zero) zero zero)
   (iszero? zero)))
Fixed typechecker

--snip--

(define (type-if pred left right)
  (if (and
       (eq? (type-expr pred) 'bool)
       (eq? (type-expr left)
            (type-expr right)))
       (type-expr left)
       (assert #f)))

--snip--
Fixed typechecker

--snip--

(define (type-if pred left right)
  (if (and
      (eq? (type-expr pred) 'bool)
      (eq? (type-expr left)
           (type-expr right)))
    (type-expr left)
    (assert #f)))

--snip--

$ racket arithmetic.rkt
verify: no counterexample found

This does not mean it's necessarily error-free, just that no counterexample was found in the search space.
Talk overview

Model checking of type checkers

Symbolic programs

Assume statements

Featherweight Java Demo

Experiments

Remaining work
The binary encoding - symbolic domain

Symbolic variable in the left node independent of root node variable.
The binary encoding - ‘zero

[Diagram of binary encoding structure]
The binary encoding - `(iszero? zero)`

Left branch node independent of root node.
A tree for Featherweight Java
Talk overview

Model checking of type checkers

Symbolic programs

Assume statements

Featherweight Java Demo

Experiments

Remaining work
What may happen during program evaluation

At runtime, a candidate program either

1. Terminates
2. Fails on run-time assertion (eg, null)
3. Diverges
4. Goes wrong (unchecked error) ← unsafe type system

We need to ignore 2 and 3. This requires assume statements which Rosette currently does not support.
Assumption by making the interpreter monadic

(define/rec (eval-expr x)
  [...])

...becomes...

(define depth 10)
(define (eval-expr x)
  (assert (> depth 0))
  (unless (> depth 0) 'depth-overflow)
  (set! depth (- depth 1))
  (define ans [...])
  (set! depth (+ depth 1))
  ans)

(define (monadic f . args)
  (if (member 'depth-overflow args)
      'depth-overflow
      (apply f args)))
Fun applications of typechecker checking

Are there programs accepted by a student’s implementation but rejected by the reference implementation?

(verify
   #:assume (student-checker program)
   #:guarantee (reference-checker program))

→ (if (iszero? zero) zero (iszero? zero))

Do all safe programs typecheck? A two-line change!

(require “hindley-milner.rkt”)
(verify-correctness
   #:typechecker interpreter
   #:interpreter typechecker)

→ (lambda (a) b)
Demo! Featherweight Java

“Classic” covariance bug:

(assert
 (is-subclass? given-arg meth-arg))

(assert
 (is-subclass? given-out meth-out))

[Note: in “real” Java this overloads methods.]
Demo! Featherweight Java
Demo! Featherweight Java

“Classic” covariance bug:

(assert
(is-subclass? given-arg meth-arg))
(assert
(is-subclass? given-out meth-out))

[Note: in “real” Java this overloads methods.]

$ racket java.rkt
Counterexample found.

class Bar extends Object {
    public Object beep(Bar X) {
        return new Main();
    }
}
class Foo extends Bar {
    public Object main(Object X) {
        return this.beep(new Bar());
    }
}
class Main extends Foo {
    public Object beep(Foo X) {
        return X.main(new Foo());
    }
}

// new Main().main(new Object())
We can explore same space with Rosette encoding

<table>
<thead>
<tr>
<th>Languages</th>
<th>Roberson et al</th>
<th>Chandra</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Program size</td>
<td>Time</td>
</tr>
<tr>
<td>Arithmetic</td>
<td>Up to 3281 AST nodes</td>
<td>38s</td>
</tr>
<tr>
<td>FJ</td>
<td>Up to 341 AST nodes</td>
<td>475s</td>
</tr>
</tbody>
</table>
Our Results

Possible to check a type system “for free” via symbolic execution

- Using just an existing typechecker and interpreter (as opposed to single-step transition)
- Using Rosette for symbolic execution (as opposed to an explicit Korat algorithm)

Two main tricks are required to make this work:

- Efficient symbolic encoding of AST data structures
- “Monadic” macros to provide assume in Rosette and bound infinite loops

Remaining work:

- Encode remaining languages from Roberson (IMP, Mini Java, Ownership Java)
- Encoding modern type systems such as Scala’s
Thanks!

Kartik Chandra, Henry M. Gunn High School
Rastislav Bodik, University of Washington