Combining Static and Dynamic Typing in Ruby

Jeff Foster
University of Maryland, College Park

Joint work with Mike Furr, David An, Mike Hicks, Mark Daly, Avik Chaudhuri, and Ben Kirzhner
Introduction

• Scripting languages are extremely popular

<table>
<thead>
<tr>
<th></th>
<th>Lang</th>
<th>Rating</th>
<th></th>
<th>Lang</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Java</td>
<td>17.9%</td>
<td>7</td>
<td>Visual Basic</td>
<td>4.5%</td>
</tr>
<tr>
<td>2</td>
<td>C</td>
<td>17.7%</td>
<td>8</td>
<td>Python</td>
<td>3.9%</td>
</tr>
<tr>
<td>3</td>
<td>C++</td>
<td>9.1%</td>
<td>9</td>
<td>Perl</td>
<td>2.4%</td>
</tr>
<tr>
<td>4</td>
<td>PHP</td>
<td>6.8%</td>
<td>10</td>
<td>JavaScript</td>
<td>2.1%</td>
</tr>
<tr>
<td>5</td>
<td>C#</td>
<td>6.7%</td>
<td>11</td>
<td>Ruby</td>
<td>1.5%</td>
</tr>
<tr>
<td>6</td>
<td>Objective-C</td>
<td>6.2%</td>
<td>12</td>
<td>Delphi</td>
<td>1.1%</td>
</tr>
</tbody>
</table>

Scripting language TIOBE Index, October 2011 (based on search hits)

• Scripting languages are great for rapid development
  ▪ Time from opening editor to successful run of the program is small
  ▪ Rich libraries, flexible syntax, domain-specific support (e.g., regexps, syscalls)
Dynamic Typing

• Most scripting languages have *dynamic typing*
  - `def foo(x)   y = x + 3; ...  # no decls of x or y`

• Benefits
  - Programs are shorter
  - No type errors unless program about to “go wrong”
  - Possible coding patterns very flexible
  - Seems good for rapid development

```java
class A {
    public static void main(String[] args) {
        System.out.println("Hello, world!");
    }
}
```

```ruby
puts "Hello, world!"
```
Drawbacks

• Errors remain latent until run time

• No static types to serve as (rigorously checked) documentation

• Code evolution and maintenance may be harder
  ▪ E.g., no static type system to ensure refactorings are type correct
Diamondback Ruby (DRuby)

• Research goal: Develop a type system for scripting langs.
  ▪ Simple for programmers to use
  ▪ Flexible enough to handle common idioms
  ▪ Provides useful checking where desired
  ▪ Reverts to run time checks where needed

• DRuby: Adding static types to Ruby
  ▪ Ruby becoming popular, especially for building web apps
  ▪ A model scripting language
    - Based on Smalltalk, and mostly makes sense internally
This Talk

• RIL: The Ruby Intermediate Language
  ▪ Small, easy to analyze subset of Ruby

• Static type inference for Ruby
  ▪ Type system is rich enough to handle many common idioms

• Profile-based analysis for highly dynamic features
  ▪ Reflection, eval, method_missing, etc

• Evaluation on a range of Ruby programs

• Extension to RubyDust (dynamic type inference)
On The Design of Ruby

• In usual language design, there are several good properties:

<table>
<thead>
<tr>
<th>consistency</th>
<th>simplicity</th>
</tr>
</thead>
<tbody>
<tr>
<td>orthogonality</td>
<td>flexibility</td>
</tr>
<tr>
<td>succinctness</td>
<td>intuitiveness</td>
</tr>
<tr>
<td>DRY (Don’t Repeat Yourself)</td>
<td>good names</td>
</tr>
<tr>
<td>generalness</td>
<td>naturalness</td>
</tr>
<tr>
<td>meets programmers’ common sense</td>
<td></td>
</tr>
</tbody>
</table>

In the design policy of Ruby, they are also good properties.

—Akira Tanaka, Language and Library API Design for Usability of Ruby, PLATEAU 2009
• However, sometimes Ruby overrides the properties for usability.
  ▪ Ruby [doesn’t] need consistency including rare usage.
  ▪ Ruby [doesn’t] need succinctness including rare usage.
  ▪ Ruby [doesn’t] need orthogonality including rare usage.
  ▪ Ruby [doesn’t] need simplicity including rare usage.
Ruby Intermediate Language (RIL)

• A front-end for Ruby code analysis and transformation

• Key features
  - GLR parser for Ruby source code
  - Compact, simplified intermediate representation
  - Pretty-printer that outputs valid, executable Ruby code
  - Partial reparsing module to make code transformation easier
  - Dataflow analysis engine
  - Support for run-time profiling
Parsing Ruby Source

- "[Ruby should] feel natural to programmers" — Yukihiro Matsumoto
  
  - Result: Grammar not amenable to LL/LR parsing
  - Ruby’s own parser is complex, written in C, tied to interpreter

```ruby
def x() return 4 end
def y()
  if false then x = 1 end
  x + 2
end
```

Local variable access!
Error: x is nil

- Solution: A GLR parser for Ruby
  
  - Grammar productions may be ambiguous
  - Ambiguities resolved eventually to yield one final parse tree
Intermediate Representation

- Ruby has many ways to do the same thing
  - if p then e / e if p / unless (not p) e / e unless (not p)

- Control flow in Ruby can be complex
  - In \( w = x().y(z()) \) does \( x() \) or \( z() \) occur first?
  - Need to know this to build flow-sensitive analyses

- Ruby has some weird behavior
  - \( x = a \)  # error if \( a \) undefined
  - if false then \( a = 3 \) end; \( x = a; \)  # sets \( x \) to nil (!)

- RIL: Simplifies this all away
  - 24 stmt kinds, each with only one side effect, organized as CFG
  - Much easier to analyze than unsimplified Ruby
Static Types for Ruby

• How do we build a static type system that accepts “reasonable” Ruby programs?
  ▪ What idioms do Ruby programmers use?
  ▪ Are Ruby programs even close to statically type safe?

• Goal: Keep the type system as simple as possible
  ▪ Should be easy for programmer to understand
  ▪ Should be predictable

• We’ll illustrate our typing discipline on the core Ruby standard library
The Ruby Standard Library

- Ruby comes with a bunch of useful classes
  - Fixnum (integers), String, Array, etc.

- However, these are implemented in C, not Ruby
  - Type inference for Ruby isn’t going to help!

- Our approach: type annotations
  - We will ultimately want these for regular code as well

- Standard annotation file base_types.rb
  - 185 classes, 17 modules, and 997 lines of type annotations
class String

###% "+" : (String) → String

###% insert : (Fixnum, String) → String

###% upto : (String) {String → Object} → String

...
end
Intersection Types

- Meth is both `Fixnum → Boolean` and `String → Boolean`
  - Ex: "foo".include?("f"); "foo".include?(42);

- Generally, if `x` has type `A` and `B`, then
  - `x` is both an `A` and a `B`, i.e., `x` is a subtype of `A` and of `B`
  - and thus `x` has both `A`'s methods and `B`'s methods

```ruby
class String
  include? : Fixnum → Boolean
  include? : String → Boolean
end
```
Intersection Types (cont’d)

- Intersection types are common in the standard library
  - 74 methods in `base_types.rb` use them
- Our types look much like the RDoc descriptions of methods
  - Except we type check the uses of functions
  - We found several places where the RDoc types are wrong
  - (Note: We treat `nil` as having any type)

```ruby
class String
  slice : (Fixnum) → Fixnum
  slice : (Range) → String
  slice : (Regexp) → String
  slice : (String) → String
  slice : (Fixnum, Fixnum) → String
  slice : (Regexp, Fixnum) → String
end
```

```ruby
str.slice(fixnum) => fixnum or nil
str.slice(fixnum, fixnum) => new_str or nil
str.slice(range) => new_str or nil
str.slice(regexp) => new_str or nil
str.slice(regexp, fixnum) => new_str or nil
str.slice(other_str) => new_str or nil
```

Element Reference—If passed a single `Fixnum`, returns the code of the character at that position. If passed two `Fixnum` objects, returns a substring.
Optional Arguments

class String
  chomp : () → String
  chomp : (String) → String
end

• Ex: “foo”.chomp(“o”); “foo”.chomp();
  ▪ By default, chomps $/$

• Abbreviation:
  0 or 1 occurrence
  class String
    chomp : (?String) → String
  end
Aside: $ in Ruby

- Global variables begin with $

- Here are all the special global variables formed from non-ascii names
  - $! $@ $; $, $/ $\ $. $\_ $< $> $$
  - $? $~ $= $\* $` $’ $+ $& $0 $: "$"
  - $1 $2 $3 $4 $5 $6 $7 $8 $9 (these are local)
Variable-length Arguments

```java
class String {
    delete : (String, *String) → String
}
```

- Ex: “foo”.delete(“a”); “foo”.delete(“a”, “b”, “c”);
- *arg is equivalent to an unbounded intersection
- To be sensible
  - Required arguments go first
  - Then optional arguments
  - Then one varargs argument

0 or more occurrences
Union Types

```ruby
class A def f() end end
class B def f() end end
x = ( if ... then A.new else B.new)
x.f
```

- This method invocation is always safe
  - Note: in Java, would make interface I s.t. A < I, B < I
- Here x has type A or B
  - It’s either an A or a B, and we’re not sure which one
  - Therefore can only invoke x.m if m is common to both A and B
- Ex: Boolean short for TrueClass or FalseClass
Structural Subtyping

• Types so far have all been nominal
  ▪ Refer directly to class names
  ▪ Mostly because core standard library is magic
    - Looks inside of `Fixnum`, `String`, etc “objects” for their contents

• But Ruby really uses structural or duck typing
  ▪ Basic Ruby op: method dispatch `e0.m(e1, ..., en)`
    - Look up `m` in `e0`, or in classes/modules `e0` inherits from
    - If `m` has `n` arguments, invoke `m`; otherwise raise error
  ▪ Most Ruby code therefore only needs objects with particular methods, rather than objects of a particular class
Object Types

module Kernel
  print : (*[to_s : () → String]) → NilClass
end

• print accepts 0 or more objects with a to_s method

• Object types are especially useful for native Ruby code:
  - def f(x) y = x.foo; z = x.bar; end

  ▪ What is the most precise type for f’s x argument?
    - Ci or C2 or ... where Ci has foo and bar methods
      - Bad: closed-world assumption; inflexible; probably does not match programmer's intention
    - Fully precise object type: [foo:() →..., bar:()→...]
Tuple Types

def f() [ 1, true ] end
a, b = f                       # a = 1, b = true

- \( f : () \rightarrow \text{Array<Fixnum or Boolean>} \)
  Not precise enough to type above example

- \( f : () \rightarrow \text{Tuple<Fixnum, Boolean>} \)
  \( \text{Tuple<}t_1, ..., t_n\text{>} = \text{array where elt } i \text{ has type } t_i \)

- Implicit subtyping between \text{Tuple} and \text{Array}
  \( \text{Tuple<}t_1, ..., t_n\text{>} < \text{Array<}t_1 \text{ or ... or } t_n\text{>} \)
That’s the Basic Type System

• Optional and varargs
• Intersection and union types
• Object types
• Tuple types
• (Plus the self type, parametric polymorphism (generics), types for mixins, first-class method types, flow-sensitivity for local variables)

• A fair amount of machinery, but not too bad!
Dynamic Features

• The basic type system works well at the application level
  ▪ Some experimental results coming up shortly

• But starts to break down if we analyze big libraries
  ▪ Libraries include some interesting dynamic features
  ▪ Typical Ruby program = small app + large libraries
class Format
  ATTRS = ["bold", "underscore", ...]
  ATTRS.each do |attr|
    code = "def #{attr}() ... end"
    eval code
  end
end
Real-World Eval Example

class Format
  ATTRS = ["bold", "underscore",...]
  ATTRS.each do |attr|
    code = "def #{attr}() ... end"
    eval code
  end
end

class Format
  def bold() ... end
  def underline() end
end
Real-World Eval Example

class Format
  ATTRS = ["bold", "underscore",...]
  ATTRS.each do |attr|
    code = "def #{attr}() ... end"
    eval code
  end
end

• **eval** occurs at top level
• **code** can be arbitrarily complex
• But, will always add the same methods
  ▪ *Morally*, this code is static, rather than dynamic
• Idea: execute the code and see what **eval** does
  ▪ Augment static analysis with this information
class Format
    ATTRS = ["bold", "underscore",...]
    ATTRS.each do |attr|
        code = "def #{attr}() ... end"
        if code = "def bold() ... end"
            def bold() ... end
        else if code = "def underscore()..."
            def underscore() ... end
        else
            safe_eval code
        end
    end
end
Profile-Guided Static Analysis

class Format
  ATTRS = ["bold", "underscore",...]
  ATTRS.each do |attr|
    code = "def #{attr}() ... end"
    if code = "def bold() ... end"
      def bold() ... end
    else if code = "def underscore()..."
      def underscore() ... end
    else
      safe_eval code
    end
  end

else case adds extra dynamic checks

Checks ensure that any runtime type error *blames* a string passed to *safe_eval*
Profiling Effectiveness

• Profiles should gather all possible strings
  ▪ Works well for code executed at the top-level
  ▪ A good test suite can capture other uses
    - Better tests yields better static checking

• Analyzed 13 benchmarks, including std-lib code
  ▪ 24,895 LOC in total

• Found five patterns
  ▪ Range from “very static” to “static enough”
  ▪ Nothing truly dynamic (e.g., no “eval $stdin.gets”)
Single String

```python
PROJECT = "SudokuSolver"
PROJECT_VERSION = eval("#{PROJECT}::VERSION")
```

- Always uses a single string
- Mimics a `#define` preprocessor directive
Collection and Bounded Set

- Values taken from a fixed collection
  - Bounded set = collection comes from non-local program points
- Single syntactic occurrence of `eval` will see many strings, but always the `same` strings
  - Mimics a macro system

```ruby
class Format
  ATTRS = ["bold", "underscore",...]
  ATTRS.each do |attr|
    code = "def #{attr}() ... end"
    eval code
  end
end
```
config = File.read(__FILE__)  
  .split(/__END__/).last  
  .gsub(#\{(.*\}\)/) { eval $1}
config = File.read(__FILE__).split(/__END__/).last
  .gsub(#\{(.*))\}/) { eval $1}

Huh?
config = File.read(__FILE__).split(/__END__/).last
  .gsub(#\{(.*))\}/) { eval $1}

class RubyForge
  RUBYFORGE_D = File.join HOME, ".rubyforge"
  COOKIE_F   = File.join RUBYFORGE_D, "cookie.dat"
  config = ...
...
end
__END__
  cookie_jar : #{ COOKIE_F }
  is_private : false
  group_ids :
    codeforpeople.com : 1024
...

Read the current file
```ruby
config = File.read(__FILE__).split(/__END__/).last
  .gsub(#\{(.*))\}(/) { eval $1}

class RubyForge
  RUBYFORGE_D = File.join HOME, ".rubyforge"
  COOKIE_F   = File.join RUBYFORGE_D, "cookie.dat"
  config = ...
...
end

__END__

cookie_jar : #{ COOKIE_F }
is_private : false
group_ids :
  codeforpeople.com : 1024
...
```

Get everything after here
config = File.read(__FILE__) 
  .split(/__END__/).last 
  .gsub(#\{(.*).\}/) { eval $1}

class RubyForge
  RUBYFORGE_D = File::join HOME, ".rubyforge"
  COOKIE_F   = File::join RUBYFORGE_D, "cookie.dat"
  config = ...
... 
end
__END__
__END__
cookie_jar : #{ COOKIE_F } 
is_private : false
group_ids :
  codeforpeople.com : 1024
...
config = File.read(__FILE__)  
  .split(/__END__/).last  
  .gsub(#\{(.*))\}/) { eval $1}

class RubyForge
  RUBYFORGE_D = File::join HOME, ".rubyforge"
  COOKIE_F   = File::join RUBYFORGE_D, "cookie.dat"
  config = ...
  ...
end

__END__

__END__

cookie_jar : #\{ COOKIE_F \}
is_private : false
group_ids : 
  codeforpeople.com : 1024
...

With this
config = File.read(__FILE__) 
  .split(/__END__/).last 
  .gsub(#\{(.*))\}/) { eval $1}

class RubyForge
  RUBYFORGE_D = File.join HOME, ".rubyforge"
  COOKIE_F   = File.join RUBYFORGE_D, "cookie.dat"
  config = ...
  ...
end
__END__
  cookie_jar : "/home/jfoster/.rubyforge/cookie.dat"
  is_private : false
  group_ids :
    codeforpeople.com : 1024
  ...

Eval it
config = File.read(__FILE__) 
  .split(/__END__/).last
  .gsub(#\{(.*))\}/) { eval $1}

class RubyForge
    RUBYFORGE_D = File::join HOME, ".rubyforge"
    COOKIE_F   = File::join RUBYFORGE_D, "cookie.dat"
    config = ...
    ...
    end

  __END__

  cookie_jar : "/home/jfoster/.rubyforge/cookie.dat"
  is_private : false
  group_ids :
    codeforpeople.com : 1024
    ...

Store in config
def test_all(obj)
    obj.methods.each do |meth|
        if meth =~ /test_/
            obj.send(meth)
        end
    end
end

class TestFoo
    def test_one ...
    def test_two ...
    end
end

• Strings are determined by the surrounding program
• Different for each client of the library
• But, fixed for the client
DRuby Architecture

I
- Require
  - main.rb

II
- All Files
- Instrument

III
- /tmp/files
- YAML

IV
- Transform
- Final Files

V
- Static Analysis
  - Runtime Stubs
• Can apply same profiling-guided analysis idea to Rails
  - Currently transformation is custom-built because
    - Profiling analysis is not context-sensitive
    - The Rails framework is really complicated; can’t analyze directly
Example Errors Found

• Typos in names
  - `Archive::Tar::ClosedStream` instead of `Archive::Tar::MiniTar::ClosedStream`
  - `Policy` instead of `Policies`

• Other standard type errors

```ruby
return rule_not_found if !@values.include?(value)
```

- `rule_not_found` not in scope
- Program did include a test suite, but this path not taken
Example Errors Found (cont’d)

- Model **Post** does not exist in the Rails app

```ruby
<% @any_more = Post.find(:first, :offset => (@offset.to_i + @posts_per_page.to_i) + 1, :limit => 1 ) %> %>

```ruby
class Integer
  def to_bn
    OpenSSL::BN.new(self)
  end
end
```

- **BN.new** expects **String**, not **Integer**
- **3.to_bn** would cause a type error
Syntactic Confusion

```
assert_nothing_raised { @hash['a', 'b'] = 3, 4 }
...
assert_kind_of(Fixnum, @hash['a', 'b'] = 3, 4)
```

- First passes `[3,4]` to the `[]=` method of `@hash`
- Second passes 3 to the `[]=` method, passes 4 as last argument of `assert_kind_of`
  - Even worse, this error is suppressed at run time due to an undocumented coercion in `assert_kind_of`
Syntactic Confusion (cont’d)

- Programmer intended to concatenate two strings
- But here the + is parsed as a unary operator whose result is discarded

```ruby
flash[:notice] = “You do not have ...“
+ “...”
```

```ruby
@count, @next, @last = 1
```

- Intention was to assign 1 to all three fields
- But this actually assigns 1 to @count, and nil to @next and @last
Performance (DRuby)

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Total LoC</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ai4r-1.0</td>
<td>21,589</td>
<td>343</td>
</tr>
<tr>
<td>bacon-1.0.0</td>
<td>19,804</td>
<td>335</td>
</tr>
<tr>
<td>hasheslice-1.0.4</td>
<td>20,694</td>
<td>307</td>
</tr>
<tr>
<td>hyde-0.0.4</td>
<td>21,012</td>
<td>345</td>
</tr>
<tr>
<td>isi-1.1.4</td>
<td>22,298</td>
<td>373</td>
</tr>
<tr>
<td>itcf-1.0.0</td>
<td>23,857</td>
<td>311</td>
</tr>
<tr>
<td>memoize-1.2.3</td>
<td>4,171</td>
<td>9</td>
</tr>
<tr>
<td>pit-0.0.6</td>
<td>24,345</td>
<td>340</td>
</tr>
<tr>
<td>sendq-0.0.1</td>
<td>20,913</td>
<td>320</td>
</tr>
<tr>
<td>StreetAddress-1.0.1</td>
<td>24,554</td>
<td>309</td>
</tr>
<tr>
<td>sudokusolver-1.4</td>
<td>21,027</td>
<td>388</td>
</tr>
<tr>
<td>text-highlight-1.0.2</td>
<td>2,039</td>
<td>2</td>
</tr>
<tr>
<td>use-1.2.1</td>
<td>20,796</td>
<td>323</td>
</tr>
</tbody>
</table>

- Times include analysis of all standard library code used by app
## Performance (DRails)

<table>
<thead>
<tr>
<th></th>
<th>LoC</th>
<th>Running times (s)</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>DRails</td>
<td>DRuby</td>
<td>Total</td>
<td></td>
</tr>
<tr>
<td>depot</td>
<td>997</td>
<td>2.30</td>
<td>9.74</td>
<td>12.04</td>
<td></td>
</tr>
<tr>
<td>moo</td>
<td>838</td>
<td>2.45</td>
<td>18.76</td>
<td>21.21</td>
<td></td>
</tr>
<tr>
<td>pubmgr</td>
<td>943</td>
<td>3.00</td>
<td>26.41</td>
<td>29.41</td>
<td></td>
</tr>
<tr>
<td>rtplan</td>
<td>1,480</td>
<td>3.47</td>
<td>26.65</td>
<td>30.12</td>
<td></td>
</tr>
<tr>
<td>amethyst</td>
<td>1,183</td>
<td>3.53</td>
<td>39.03</td>
<td>42.56</td>
<td></td>
</tr>
<tr>
<td>diamondlist</td>
<td>1,415</td>
<td>4.10</td>
<td>23.81</td>
<td>27.91</td>
<td></td>
</tr>
<tr>
<td>chucklist</td>
<td>1,447</td>
<td>4.08</td>
<td>52.23</td>
<td>56.31</td>
<td></td>
</tr>
<tr>
<td>boxroom</td>
<td>2,330</td>
<td>4.16</td>
<td>87.23</td>
<td>91.39</td>
<td></td>
</tr>
<tr>
<td>onyx</td>
<td>2,228</td>
<td>5.62</td>
<td>79.75</td>
<td>85.37</td>
<td></td>
</tr>
<tr>
<td>mystic</td>
<td>2,822</td>
<td>6.38</td>
<td>146.40</td>
<td>152.78</td>
<td></td>
</tr>
<tr>
<td>lohimedia</td>
<td>11,106</td>
<td>14.01</td>
<td>662.95</td>
<td>676.96</td>
<td></td>
</tr>
</tbody>
</table>
Take Two: Rubydust

• **Ruby Dynamic Unraveling of Static Types**
• Pure Ruby library
  ▪ Dynamic analysis--does not examine source code
• Runs test cases under Rubydust
  ▪ Generates type constraints, solves the constraints, and reports type errors and inferred types
• Takes advantage of the test cases that come with Ruby programs
Rubydust is Effective

• Avoids need for parser, IR, precise semantics spec.
• Sound if every path in each method is covered
  ▪ Sound = types valid for all possible runs
• Inferred types are useful (even if not all paths covered)
  ▪ Readable, intuitive, available for introspection
• Application of our approach to eight programs had valid types, all of their types were sound, readable
  ▪ Another program had a type error we found
Rubydust: Dynamic Type Inference

• Idea: take advantage of test first culture and piggyback type inference on top of testing

• Works in three steps:
  1. Dynamically instrument Ruby program
  2. Run the program, generating constraints
  3. When the runs are complete, solve these constraints to produce static types
Type Variables and Wrapping

- Each method in a class is assigned a type variable for each of its arguments and return value
  - `class C def bar(x) ... end end`
  - has type `bar: αC_bar_x → αC_bar_ret`
- Rubydust instrumentation wraps values with proxies
  - `v : α` denotes a value `v` wrapped with a type variable `α`
  - Type variable expresses a static invariant on run-time values
- Wrapping done at each method call and return
  - Type variable determined by the method actually called
def bar(x)
    x.qux()
    return 7
end

y = bar(C.new)
def bar(x):
    x.qux()
    return 7
end
y = bar(C.new)

Constraints:

An instance of C

objc
**Wrapping for Constraint Generation**

```python
def bar(x)
    x.quux()
    return 7
end

y = bar(C.new)
```

**Constraints:**

- \( C \leq \alpha x \)
- \( \alpha x \leq \text{bar_ret} \)

\[ \text{obj}_c \]
def bar(x)
    x.qux()
    return 7
end

y = bar(C.new)

Constraints:
typeof(x) ≤ α_x

x now points to obj_c
def bar(x):
    x.qux()
    return 7
end

y = bar(C.new)

Constraints:
\[ \text{typeof}(x) \leq \alpha_x \]

generate a constraint for the argument \( x \)

\( x \) \( \text{objc} \)
def bar(x):
    x.qux()
    return 7
end

y = bar(C.new)

Constraints:

\[ C \leq \alpha_x \]
def bar(x)
    x.qux()
    return 7
end

y = bar(C.new)

Constraints:
\[ C \leq \alpha_x \]

\( x \) \rightarrow \text{wrapping obj}_{c} \text{ with } \alpha_x
def bar(x):
    x.qux()
    return 7
end

y = bar(C.new)

Constraints:
C ≤ α_x

x

(objc : α_x)
def bar(x):
    x.qux()
    return 7
end
y = bar(C.new)

Constraints:

- \( C \leq \alpha_x \)
- \( \text{typeof}(x) \leq [\text{qux} : () \rightarrow ()] \)

\( x \) (\( \text{objc} : \alpha_x \))

**generate a constraint for the receiver** \( x \)
def bar(x)
    x.qux()
    return 7
end

y = bar(C.new)

Constraints:

\[ C \leq \alpha_x \]

\[ \alpha_x \leq [\text{qux: ()} \rightarrow ()] \]
def bar(x)
    x.qux()
    return 7
end

ty = bar(C.new)

Constraints:
C \leq \alpha_x
\alpha_x \leq [\text{qux: ()} \rightarrow ()]

x

(\text{objc : } \alpha_x)
def bar(x)
    x.qux()
    return 7
end

y = bar(C.new)

Constraints:
C ≤ α_x
α_x ≤ [qux: () ➝ ()]

x

(objc : α_x) 7
def bar(x):
    x.qux()
    return 7
end

y = bar(C.new)

Constraints:

\[
C \leq \alpha_x
\]
\[
\alpha_x \leq [\text{qux}: () \rightarrow ()]
\]
\[
\text{typeof}(7) \leq \alpha_{\text{bar_ret}}
\]
def bar(x)
    x.qux()
    return 7
end

y = bar(C.new)

Constraints:

C \leq \alpha_x

\alpha_x \leq [\text{qux: } () \rightarrow ()]

\text{Numeric} \leq \alpha_{\text{bar\_ret}}
def bar(x)
    x.qux()
    return 7
end

y = bar(C.new)

Constraints:

\[
C \leq \alpha_x \\
\alpha_x \leq \{\text{qux: () \to ()}\} \\
\text{Numeric} \leq \alpha_{\text{bar_ret}}
\]

\[
\text{bar_ret}
\]

\[
\text{(objc : } \alpha_x) \\
7
\]
def bar(x)
    x.qux()
    return 7
end

y = bar(C.new)

Constraints:
C ≤ \alpha_x
\alpha_x ≤ [qux: () \rightarrow ()]
Numeric ≤ \alpha_{\text{bar_ret}}
def bar(x)
    x.qux()
    return 7
end

y = bar(C.new)

Constraints:

\[ C \leq \alpha_x \]
\[ \alpha_x \leq [\text{qux}: () \rightarrow ()] \]
\[ \text{Numeric} \leq \alpha_{\text{bar\_ret}} \]

\((\text{objc} : \alpha_x)\)

\((7 : \alpha_{\text{bar\_ret}})\)
Wrapping and Constraint Generation Summary

\[
\text{def } m(x) \ e \ \text{end; } \ o.m(v)
\]

1. Constrain arguments: \( \text{typeof}(v) \leq \alpha_x \)
2. Wrap arguments: \((v : \alpha_x)\)
3. Evaluate method body: \(e \Rightarrow v'\)
4. Constrain return: \(\text{typeof}(v') \leq \alpha_{m\_ret}\)
5. Wrap return: \((v' : \alpha_{m\_ret})\)
6. (We handle fields similarly; details in our paper)
Constraint Resolution

Goal: find least restrictive types

Constraints:
\[ C \leq \alpha_x \]
\[ \alpha_x \leq [\text{qux}: () \rightarrow ()] \]
\[ \text{Numeric} \leq \alpha_{\text{bar_ret}} \]

Signature:
\[ \text{bar}: \alpha_x \rightarrow \alpha_{\text{bar_ret}} \]
Constraint Resolution

Constraints:

\[ C \leq \alpha_x \]
\[ \alpha_x \leq [\text{qux: () } \rightarrow ()] \]
\[ \text{Numeric } \leq \alpha_{\text{bar_ret}} \]
\[ C \leq [\text{qux: () } \rightarrow ()] \]

Signature:

\[ \text{bar: } \alpha_x \rightarrow \alpha_{\text{bar_ret}} \]
Constraint Resolution

Constraints:

\[ C \leq \alpha_x \]
\[ \alpha_x \leq [\text{qux: () } \rightarrow ()] \]

**Numeric \leq \alpha_{\text{bar_ret}}**

\[ C \leq [\text{qux: () } \rightarrow ()] \]

Signature:

\[ \text{bar: } \alpha_x \rightarrow \alpha_{\text{bar_ret}} \]
Constraint Resolution

Constraints:

\[ \alpha_x \leq \text{Numeric} \leq \text{Numeric} \]

Signature:

\[ \text{bar: } \alpha_x \rightarrow \text{Numeric} \]
Constraint Resolution

Constraints:
\[ C \leq \alpha_x \]
\[ \alpha_x \leq [\text{qux}: () \rightarrow ()] \]
\[ \text{Numeric} \leq \text{Numeric} \]
\[ C \leq [\text{qux}: () \rightarrow ()] \]

Signature:
\[ \text{bar}: \alpha_x \rightarrow \text{Numeric} \]
Constraint Resolution

Constraints:

\[ C \leq \alpha_x \]

\[ \alpha_x \leq \text{[qux: ()} \rightarrow ()] \]

\[ \text{Numeric} \leq \text{Numeric} \]

\[ C \leq \text{[qux: ()} \rightarrow ()] \]

Signature:

\[ \text{bar: } \alpha_x \rightarrow \text{Numeric} \]

For contravariant args, find least upper bound
Constraint Resolution

Constraints:

\[ C \leq \alpha_x \]

\[ \text{Numeric} \leq \text{Numeric} \]

\[ C \leq [\text{qux:()} \rightarrow ()] \]

Signature:

\[ \text{bar: [qux:()} \rightarrow ()] \rightarrow \text{Numeric} \]
Constraint Resolution

Constraints:

\[ C \leq \alpha_x \]
\[ [\text{qux:()} \rightarrow ()] \leq [\text{qux:()} \rightarrow ()] \]
\[ \text{Numeric} \leq \text{Numeric} \]
\[ C \leq [\text{qux:()} \rightarrow ()] \]

Signature:

\[ \text{bar: [qux:()} \rightarrow ()] \rightarrow \text{Numeric} \]
Constraints:

\[ C \leq \alpha_x \]
\[ [\text{qux:()} \rightarrow ()] \leq [\text{qux:()} \rightarrow ()] \]

Numeric \leq Numeric

\[ C \leq [\text{qux:()} \rightarrow ()] \]

Signature:

\[ \text{bar: [qux:()} \rightarrow ()] \rightarrow \text{Numeric} \]
Benefit: Path Sensitivity

- Static type inference would model infeasible paths

```python
def bar(p)
    if p then y = 3 else y = "hello" end
    if p then y + 6 else y.length end
end
```
Benefit: Path Sensitivity

- Static type inference would model infeasible paths

```python
def bar(p):
    if p then y = 3 else y = "hello" end
    if p then y + 6 else y.length end
end
```

\[
\text{Numeric} \leq \alpha_y
\]
• Static type inference would model infeasible paths

```python
def bar(p):
    if p then y = 3 else y = "hello" end
    if p then y + 6 else y.length end
end
```

\[
\text{Numeric} \leq \alpha_y
\]

\[
\text{String} \leq \alpha_y
\]
Benefit: Path Sensitivity

- Static type inference would model infeasible paths

```python
def bar(p):
    if p then y = 3 else y = “hello” end
    if p then y + 6 else y.length end
end
```

\[
\text{Numeric} \leq \alpha_y \\
\text{String} \leq \alpha_y \\
\alpha_y \leq [\text{Numeric} \rightarrow \beta]
\]
Benefit: Path Sensitivity

- Static type inference would model infeasible paths

```python
def bar(p):
    if p then y = 3 else y = "hello" end
    if p then y + 6 else y.length end
end
```

\[
\begin{align*}
\text{Numeric} & \leq \alpha_y \\
\text{String} & \leq \alpha_y \\
\alpha_y & \leq [+ : \text{Numeric} \rightarrow \beta] \\
\alpha_y & \leq [\text{length: ()} \rightarrow \gamma]
\end{align*}
\]
Benefit: Path Sensitivity

- Static type inference would model infeasible paths

```python
def bar(p):
    if p then y = 3 else y = “hello” end
    if p then y + 6 else y.length end
end
```

\[
\begin{align*}
\text{Numeric} & \leq \alpha_y \\
\text{String} & \leq \alpha_y \\
\alpha_y & \leq [+ : \text{Numeric} \rightarrow \beta] \\
\alpha_y & \leq [\text{length: ()} \rightarrow \gamma]
\end{align*}
\]
Benefit: Path Sensitivity

• Static type inference would model infeasible paths

```python
def bar(p)
    if p then y = 3 else y = "hello" end
    if p then y + 6 else y.length end
end
```

```
Numeric ≤ α_y
String ≤ α_y
α_y ≤ [+ : Numeric → β]
α_y ≤ [length: () → γ]
```

Unsatisfiable (but infeasible)
Benefit: Path Sensitivity

- Rubydust observes actual executions -- more precise

```ruby
def bar(p)
  if p then y = 3 else y = "hello" end
  if p then y + 6 else y.length end
end
```

No “local” variables $\alpha_x, \alpha_y$

$\text{String} \leq [\text{length: } () \rightarrow \text{Numeric}]$
Benefit: Path Sensitivity

• Rubydust observes actual executions -- more precise

def bar(p)
    if p then y = 3 else y = “hello” end
    if p then y + 6 else y.length end
end

String ≤ [length: () → Numeric]
Numeric ≤ [+ : Numeric → ...]
class Format
  ATTRS = ["bold", "underscore", ...]
  ATTRS.each do |attr|
    code = "def #{attr}() ... end"
    eval code
  end
end
class Format
  ATTRS = ['bold', 'underscore', ...]
  ATTRS.each do |attr|
    code = "def #{attr}() ... end"
    eval code
  end
end
Observes result of eval

```ruby
class Format
  ATTRS = ["bold", "underscore", ...]
  ATTRS.each do |attr|
    code = "def #{attr}() ... end"
    eval code
  end
end
```

- **eval** occurs at top level
- **code** can be arbitrarily complex
- But, will always add the same methods
  - *Morally*, this code is static, rather than dynamic
- Rubydust instruments what eval produces
Soundness

- **Theorem**: a method’s type is sound for all executions that follow previously observed paths

- **Corollary**: we are sound if we have observed all program paths for each method
Path Coverage

• Only need paths within a method body, not all possible interactions between methods

```python
def bar(y, z):
    if y then foo(z) else foo(!z)
end
def foo(x):
    if x then 0 else “hello”
end
```
Path Coverage

• Only need paths within a method body, not all possible interactions between methods

```python
def bar(y,z):
    if y then foo(z) else foo(!z)
end
def foo(x):
    if x then 0 else "hello"
end
bar(true,true)
```

![Diagram of path coverage](https://via.placeholder.com/150)
Path Coverage

• Only need paths within a method body, not all possible interactions between methods

```python
def bar(y,z):
    if y then foo(z) else foo(!z)
end

def foo(x):
    if x then 0 else "hello"
end

bar(true,true)
bar(false,true)
```

```python
def bar(y,z):
    if y then foo(z) else foo(!z)
end

def foo(x):
    if x then 0 else "hello"
end

bar(true,true)
bar(false,true)
```
Path Coverage

• Only need paths within a method body, not all possible interactions between methods

```
def bar(y,z)
    if y then foo(z) else foo(!z)
end
def foo(x)
    if x then 0 else “hello”
end
bar(true,true)
bar(false,true)
```

Sufficient for soundness
Path Coverage

• Only need paths within a method body, not all possible interactions between methods

```plaintext
def bar(y,z)
    if y then foo(z) else foo(!z)
end
def foo(x)
    if x then 0 else "hello"
end
bar(true,true)
bar(false,true)
bar(true,false)
```

Redundant
Path Coverage

• Only need paths within a method body, not all possible interactions between methods

```python
def bar(y,z):
    if y then foo(z) else foo(!z)
end

def foo(x):
    if x then 0 else “hello”
end

bar(true,true)
bar(false,true)
bar(true,false)
bar(false,false)
```
Implementation

Ruby Execution
Implementation

Program

Annotation Parser

Instrumentation

Testing

Constraints

Solver

Types

Rubydust Execution
Instrumentation

class A

foo
bar
baz
...

t(v) ≤ α_x

t(v') ≤ α_{foo\_ret}
Instrumentation

class A

foo
bar
baz
...

t(v) \leq \alpha_x

\quad

t(v') \leq \alpha_{\text{foo\_ret}}
Instrumentation

```ruby
class A
  ___foo
  ___bar
  ___baz
  ...

\[ t(v) \leq \alpha_x \]
\[ t(v') \leq \alpha_{\text{foo\_ret}} \]
```
Instrumentation

class A

method missing

___foo
___bar
___baz
...

\[ t(v) \leq \alpha_x \]
\[ t(v') \leq \alpha_{\text{foo_ret}} \]
Instrumentation

```
class A

foo(v)

method missing

___foo
___bar
___baz
...

\[ t(v) \leq \alpha_x \]
\[ t(v') \leq \alpha_{\text{foo_ret}} \]
```
Instrumentation

class A

foo(v)

v

method
missing

__foo
__bar
__baz
...

t(v) \leq \alpha_x

t(v') \leq \alpha_{\text{foo_ret}}
Instrumentation

class A

foo(v)

v

method missing

v : $\alpha_x$

___foo

___bar

___baz

...

t(v) \leq \alpha_x

t(v') \leq \alpha_{\text{foo_ret}}
class A

foo(v)

method missing

__foo
__bar
__baz
...

t(v) ≤ α_x

t(v') ≤ α_{foo_ret}
Instrumentation

\[
\text{method missing}
\]

\[
\text{foo(v)}
\]

\[
\text{\(v\)' : } \alpha_{\text{foo_ret}}
\]

\[
\text{\(t(v) \leq \alpha_x\)}
\]

\[
\text{\(t(v') \leq \alpha_{\text{foo_ret}}\)}
\]
Limitations

• Block types not inferred
• Annotation/inference per-class, not per method
• Inefficient handling of arrays and hashes
• No handling of dynamic method creation (after instrumentation)
• No handling of global variables or constants
• No polymorphic or intersection type inference
• Slow solver, slow instrumentation
Experiments

• Ran eight small programs under Rubydust
  ▪ Used test suites that came with the programs

• Ran on a 2.5Ghz dual core processor with 4GB memory running Mac OS X (Snow Leopard)

• Produced sound types for seven programs

• Found one type error
# Implementation and Experiments

**Ruby library** *(reflect and self-instrument)*  
+ *type annotations for standard C libraries*

<table>
<thead>
<tr>
<th>Library</th>
<th>LOC</th>
<th>Tests</th>
<th>Paths</th>
<th>Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ministat</td>
<td>96</td>
<td>10</td>
<td>19</td>
<td>84%</td>
</tr>
<tr>
<td>Finitefield</td>
<td>103</td>
<td>9</td>
<td>14</td>
<td>93%</td>
</tr>
<tr>
<td>Ascii85</td>
<td>105</td>
<td>7</td>
<td>67</td>
<td>28%</td>
</tr>
<tr>
<td>A-star</td>
<td>134</td>
<td>1</td>
<td>41</td>
<td>62%</td>
</tr>
<tr>
<td>Hebruby</td>
<td>178</td>
<td>19</td>
<td>36</td>
<td>91%</td>
</tr>
<tr>
<td>Style</td>
<td>237</td>
<td>12</td>
<td>88</td>
<td>28%</td>
</tr>
<tr>
<td>Rubyk</td>
<td>258</td>
<td>1</td>
<td>37</td>
<td>68%</td>
</tr>
<tr>
<td>StreetAddress</td>
<td>772</td>
<td>1</td>
<td>23</td>
<td>88%</td>
</tr>
</tbody>
</table>
## Results

<table>
<thead>
<tr>
<th>Package</th>
<th>LOC</th>
<th>TC</th>
<th>E(#)</th>
<th>LCover</th>
<th>MCover</th>
<th>P(#)</th>
<th>PCover</th>
<th>OT(s)</th>
<th>RT(s)</th>
<th>Solving(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ministat-1.0.0</td>
<td>96</td>
<td>10</td>
<td>7</td>
<td>75%</td>
<td>11 / 15</td>
<td>19</td>
<td>84%</td>
<td>0.00</td>
<td>11.19</td>
<td>57.11</td>
</tr>
<tr>
<td>finitefield-0.1.0</td>
<td>103</td>
<td>9</td>
<td>6</td>
<td>98%</td>
<td>12 / 12</td>
<td>14</td>
<td>93%</td>
<td>0.00</td>
<td>1.74</td>
<td>1.28</td>
</tr>
<tr>
<td>Ascii85-1.0.0</td>
<td>105</td>
<td>7</td>
<td>3</td>
<td>95%</td>
<td>2 / 2</td>
<td>67</td>
<td>28%</td>
<td>0.01</td>
<td>6.81</td>
<td>0.17</td>
</tr>
<tr>
<td>a-star</td>
<td>134</td>
<td>1</td>
<td>5</td>
<td>100%</td>
<td>20 / 24</td>
<td>41</td>
<td>62%</td>
<td>0.04</td>
<td>114.81</td>
<td>37.46</td>
</tr>
<tr>
<td>hebruby-2.0.2</td>
<td>178</td>
<td>19</td>
<td>8</td>
<td>81%</td>
<td>20 / 26</td>
<td>36</td>
<td>91%</td>
<td>0.01</td>
<td>19.97</td>
<td>19.08</td>
</tr>
<tr>
<td>style-0.0.2</td>
<td>237</td>
<td>12</td>
<td>1</td>
<td>75%</td>
<td>17 / 32</td>
<td>88</td>
<td>28%</td>
<td>0.01</td>
<td>8.46</td>
<td>0.28</td>
</tr>
<tr>
<td>Rubyk</td>
<td>258</td>
<td>1</td>
<td>4</td>
<td>69%</td>
<td>15 / 20</td>
<td>37</td>
<td>68%</td>
<td>0.00</td>
<td>7.33</td>
<td>0.56</td>
</tr>
<tr>
<td>StreetAddress-1.0.1</td>
<td>772</td>
<td>1</td>
<td>10</td>
<td>79%</td>
<td>33 / 44</td>
<td>23</td>
<td>88%</td>
<td>0.02</td>
<td>4.45</td>
<td>24.58</td>
</tr>
</tbody>
</table>

TC - test cases  
E - manual edits  
LCover - line coverage  
MCover - method coverage / total # of methods  
P - paths  
PCover - path coverage  
OT - original running time  
RT - Rubydust running time
Structural Types

class MiniStat::Data
...

typesig ("median : ([ sort! : () → Array<Numeric>;
    size : () → Numeric;
    '['] : (Numeric) → Numeric;
    '==' : (Object) → Boolean]
    → Numeric ")
...
end
class MiniStat::Data

... typesig ("median : ([ sort! : () \rightarrow Array<Numeric>; size : () \rightarrow Numeric;
    ['[]' : (Numeric) \rightarrow Numeric;
    '==' : (Object) \rightarrow Boolean])
    \rightarrow Numeric ")

...
end

Requires these 4 methods
class FiniteField

... typesig("binary_mul : ([
    ‘&’ : (Numeric) → Numeric;
    ‘≫’ : (Numeric) → Numeric;
    ‘==’ : (Object) → Boolean,
    Numeric)
    → Numeric ”)
...

... typesig("inverse : ([‘>’ : (Numeric) → Boolean;
    ‘≪’ : (Numeric) → Numeric;
    ‘&’ : (Numeric) → Numeric;
    ‘≫’ : (Numeric) → Numeric;
    ‘==’ : (Object) → Boolean;
    ‘^’ : (Numeric) → Numeric])
    → Numeric ”)
module Ascii85
  def self.encode(str, wrap_lines = 80)
    ...
    if (!wrap_lines)
      return ...
    end
    ...
    ... wrap_lines.to_i
    end
    ...
  end
module Ascii85
  def self.encode(str, wrap_lines = 80)
    ...
    if (!wrap_lines)
      return ...
    end
    ...
    ... wrap_lines.to_i
    end
  end
end
module Ascii85
  def self.encode(str, wrap_lines = 80)
    ...
    # return from the method if false
    if (!wrap_lines)
      return ...
    end
    ...
    ... wrap_lines.to_i
    end
  end
end
module Ascii85
  def self.encode(str, wrap_lines = 80)
    ...
    if (!wrap_lines)
      return ...
    end
    ...
    wrap_lines.to_i
  end
  ...
end
module Ascii85
  def self.encode(str, wrap_lines = 80)
    ...
    if (!wrap_lines)
      return ...
    end
    ...
    ...
    wrap_lines.to_i
  end
  end

[ERROR] Boolean !<: [to_i : () → Numeric]
## Comparison

<table>
<thead>
<tr>
<th>Static inference</th>
<th>Dynamic inference</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Sound</td>
<td>• Sound</td>
</tr>
<tr>
<td>▪ no test cases required</td>
<td>▪ requires test cases</td>
</tr>
<tr>
<td>• Less expressive</td>
<td>• More expressive</td>
</tr>
<tr>
<td>▪ flow insensitive</td>
<td>▪ locally path sensitive</td>
</tr>
<tr>
<td>▪ no dynamic features</td>
<td>▪ some dynamic features</td>
</tr>
<tr>
<td>• Less scalable</td>
<td>• No code-size wall</td>
</tr>
<tr>
<td>▪ whole program analysis</td>
<td>▪ instrumentation overhead</td>
</tr>
<tr>
<td>• More infrastructure</td>
<td>• Reuses existing facilities</td>
</tr>
<tr>
<td>▪ parser, semantics</td>
<td>▪ builds on reflection</td>
</tr>
</tbody>
</table>
Future Work

• Faster implementation
• Inference of polymorphic and intersection types
• Relax “load time” / “run-time” barrier for instrumentation
• Apply to Ruby on Rails