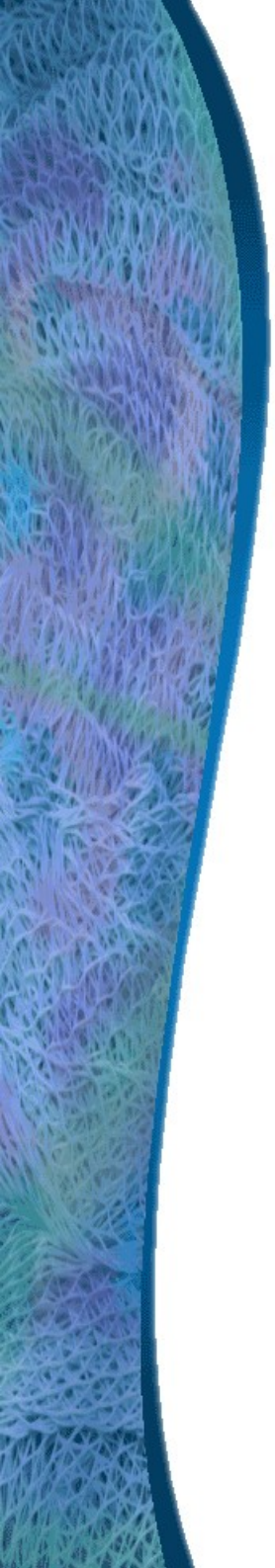


Static Type Inference for Ruby

Based on work of M. Furr et al.
University of Maryland

Sander Sõnajalg

Agenda

- Motivation
- Ruby Intermediate Language (RIL)
- Type annotations for standard library
- Type inference system

The Ruby Language

```ruby
class Person
  @@number_of_persons = 0
  attr_accessor :name, :age
  def initialize(name, age)
    @name, @age = name, age
    @@number_of_persons += 1
  end
  def to_s
    "#@name (#@age)"
  end
  def apply_block &block
    block.call(self)
  end
end

a = Person.new("Edgar", 23)
a.age = 24

increase_age = proc {|p| p.age = p.age + 20}
a.apply_block &increase_age
```

CLASS VARIABLE

CODE GENERATION

CONSTRUCTOR

INSTANCE VARIABLE

CLOSURE PASSING AND EVALUATION

CLOSURE CREATION

The Problem

- Ruby is a dynamically typed language
- We would like to know the types statically
- .. without explicitly annotating them

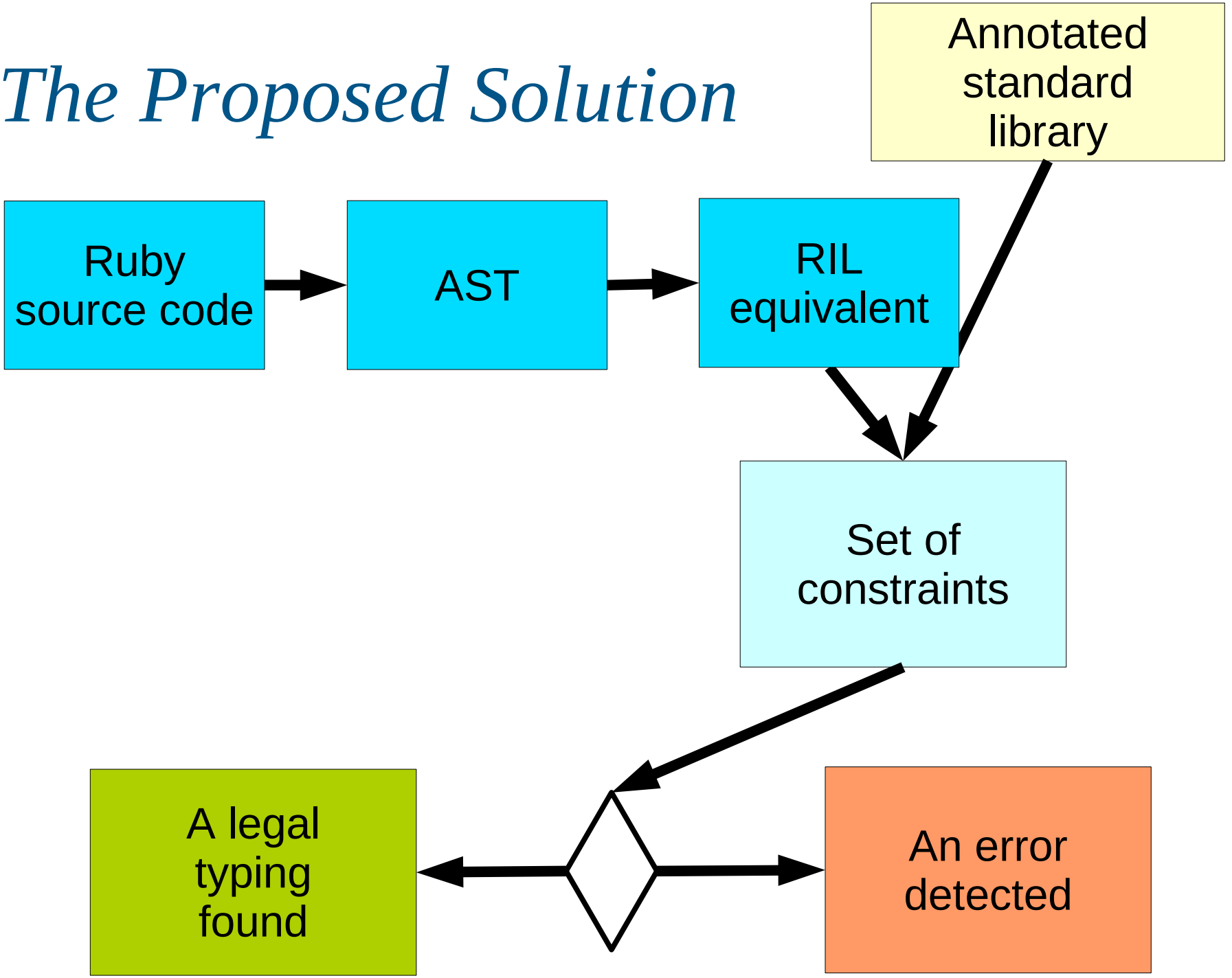
```
def foo
  a = String.new
  a << "foo"
  return a
end

(String.new).length

(String.new).bar
```

OK,
returns String

OK,
returns Fixnum

Not OK,
NoMethodError

The Proposed Solution



Example 0

```
def foo
  return "bar"
end
```

Ruby Intermediate Language (RIL)

- A subset of ruby
- Syntactic sugar removed

(1) if p then e end
(2) unless (not p) then e end

(3) e if p
(4) e unless (not p)

Type Annotations Language

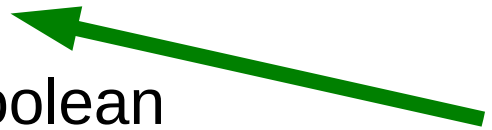
$A \in$ *class and module names*
$m \in$ *method names*
$t \in$ *type variables*

$$
\begin{array}{lll}
typ & ::= & A\langle typ_1, \ldots \rangle \mid obj\_typ \mid typ \textbf{ or } typ \mid t \\
obj\_typ & ::= & [m_0 : meth\_typ_0, \ldots] \\
meth\_typ & ::= & (param\_typs)[\{meth\_typ\}]^? \rightarrow typ \\
param\_typs & ::= & typ_1, \ldots, [?typ_n, \ldots]^?, [*typ]^? \\
typ\_decl & ::= & m\langle t_1, \ldots \rangle : meth\_typ
\end{array}
$$

Type Annotations: Examples

```
class String
  ”+” : (String) → String
  insert : (Fixnum, String) → String
  upto : (String) {String → Object}→ String
  chomp : (?String) → String

  delete : (String , ∗String) → String

  include? : Fixnum → Boolean
  include? : String → Boolean

  # ..
end
```

OPTIONAL ARGUMENT



VARARGS

Type Annotations: Examples (2)

OBJECT TYPE
(STRUCTURAL TYPE)

```
module Kernel
  print : (∗[ to s : () → String]) → NilClass
  clone : () → self
end
```

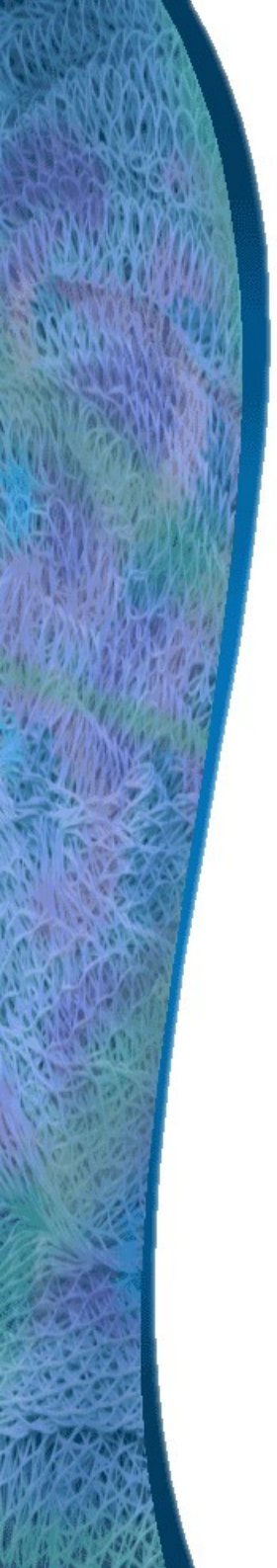
SELF TYPE

Type Annotations: Examples (3)

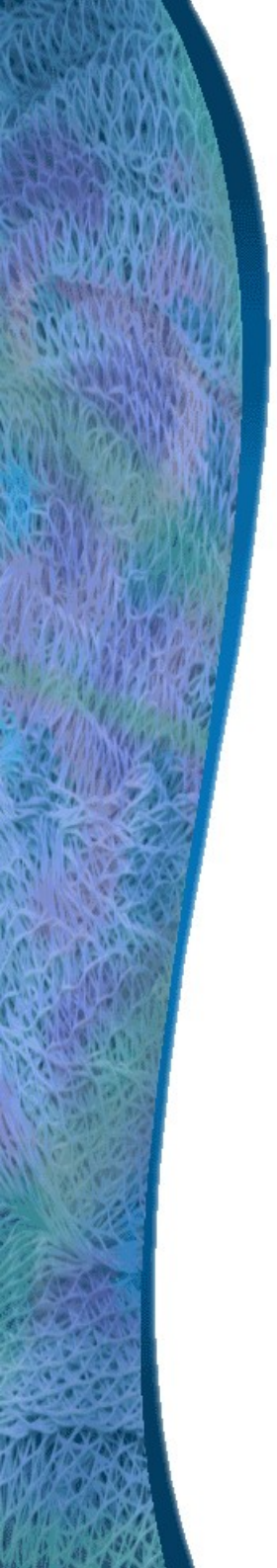
PARAMETRIC POLYMORPHISM

```
class Array<t>
  at : (Fixnum) → t

  first : () → t
  first : (Fixnum) → Array<t>

  collect <u> : () {t → u} → Array<u>
  "+"<u> : (Array<u>) → Array<t or u>
end
```

UNION TYPE



Type Inference System: The Types



Type Inference System: The Types (2)

Type Inference System: Environments and Contexts

Γ – an environment; bindings for local variables

Ω – current context; hold type of the current class

- Ω.cls – type of current class
- Ω.blk – type of current block
- Ω.ret – current return type

Type Inference System: Constraints

- $\tau \leq \tau'$
- $bt1 \leq bt2$
- $mt1 \leq mt2$
- $\tau \Rightarrow \tau'$
- $\tau \in \tau'.par$
- $(m : mt) \in \tau.mths$
- $A : \tau$
- $x : \tau$

Type Inference

- $\mathbf{C}; \Gamma; \Omega \vdash \mathbf{e} : \tau$
- $\mathbf{C}; \Gamma; \Omega \vdash \mathbf{s}; \Gamma_2$
- $C \vdash C_2$

Type Inference: Expressions

(NIL)

$$\frac{}{C;\Gamma;\Omega \vdash_e \mathsf{nil} : Nom_{\mathsf{nil}}}$$

(INT)

$$\frac{}{C;\Gamma;\Omega \vdash_e n : Nom_{\mathsf{Fixnum}}}$$

(LOCAL/SELF)

$$\frac{id \in \mathrm{dom}(\Gamma) \quad id \text{ not captured}}{C;\Gamma;\Omega \vdash_e id : \Gamma(id)}$$

(FIELD)

$$\frac{C \vdash \Gamma(\mathsf{self}) \leq [@x : \alpha] \qquad \alpha \text{ fresh}}{C;\Gamma;\Omega \vdash_e @x : \alpha}$$

(TUPLE)

$$\frac{C;\Gamma;\Omega \vdash_e e_i : \tau_i \qquad i \in 1..n}{C;\Gamma;\Omega \vdash_e [e_1, \ldots, e_n] : (\tau_1 \times \ldots \times \tau_n)}$$

Type Inference: Statements

(SEQ)

$$\frac{C;\Gamma,\Omega \vdash s_1;\Gamma' \quad C;\Gamma';\Omega \vdash s_2;\Gamma''}{C;\Gamma;\Omega \vdash s_1;s_2;\Gamma''}$$

(IF)

$$\frac{C;\Gamma;\Omega \vdash_e e_1 : \tau \qquad C;\Gamma;\Omega \vdash s_2;\Gamma_2 \qquad C;\Gamma;\Omega \vdash s_3;\Gamma_3}{C;\Gamma;\Omega \vdash \text{if } e_1 \text{ then } s_2 \text{ else } s_3;\Gamma_2 \cup \Gamma_3}$$

(LASSIGN)

$$\frac{C;\Gamma;\Omega \vdash_e e : \tau \quad \Gamma' = \Gamma[x \mapsto \tau] \quad x \text{ not captured}}{C;\Gamma;\Omega \vdash x = e;\Gamma'}$$

(CASSIGN)

$$\frac{C;\Gamma;\Omega \vdash_e e : \tau \quad C \vdash A : \tau}{C;\Gamma;\Omega \vdash A = e;\Gamma}$$

Type Inference: Statements (2)

(NEW)

$$\frac{C;\Gamma;\Omega \vdash_e e : \tau \qquad C \vdash \tau \Rightarrow \alpha \qquad \alpha \text{ fresh} \qquad \Gamma' = \Gamma[x \mapsto \alpha]}{C;\Gamma;\Omega \vdash x = e.\mathsf{new};\Gamma'}$$

(RETURN)

$$\frac{C;\Gamma,\Omega \vdash_e e : \tau \qquad C \vdash \tau \leq \Omega.\mathsf{ret}}{C;\Gamma;\Omega \vdash \mathsf{return}\ e, \Gamma}$$

(CALL)

$$\frac{C;\Gamma;\Omega \vdash_e e_i : \tau_i \quad i \in 0..n \quad C;\Gamma;\Omega \vdash_e b : bt \qquad C \vdash \tau_0 \leq [m : (\tau_0 \times \cdots \times \tau_n \times bt) \to \alpha] \qquad \Gamma' = \Gamma[x \mapsto \alpha] \qquad \alpha \text{ fresh}}{C;\Gamma;\Omega \vdash x = e_0.m(e_1,\ldots,e_n)b;\Gamma'}$$

Example 1

```
def m arg
  if arg == 0
    a = "STRING"
  else
    a = 12345
  end
  return a
end
```

Example 2

```
class B
  def id arg
    return arg
  end
end

y = B.new
x = y.id(123)
```

Example 3

```
def foo
  a = String.new
  a.sgioasga()
end
```

Results

Program	LOC	Changes	Tm (s)	FPos
merge-bibtex	103	None	2.0	0
sudokusolver-1.4	152	None	3.3	35
ObjectGraph-1.0.1	153	None	2.3	1
itcf-1.0.0	178	S-2	4.3	0
text-highlight-1.0.2	283	S-1, M-2	2.7	1
rhotoalbum-0.4	313	S-2	3.5	0
gs_phone-0.0.4	542	S-1	2.6	0
StreetAddress-1.0.1	892	(S, R)-1	36.1	0

S–added stub file; M–manually expanded meta-programming code;
R–replaced require argument with constant String

Limitations

- Types that cannot be described
  - Array.flatten:
    - [[[1,2],3, 4], 5, 6] => [1, 2, 3, 4, 5, 6]
- Class definitions changed at runtime
- eval

```
ATTRIBUTES.each do |attr|
  code = "def set_#{attr}(value)
                  #{attr} = value; end"
  eval code
end
```