Concrete Partial Evaluation in Ruby
Andrew Keep  Arun Chauhan
Department of Computer Science
Indiana University
{akeep,achauhan}@cs.indiana.edu

Introduction
Modern scientific research is a collaborative process, with researchers from many disciplines and institutions working toward a common goal. Dynamic languages, like Ruby, provide a platform for quickly developing simulation and analysis tools, freeing researchers to focus on research instead of developing infrastructure. Ruby fits this role well, allowing:

- Incorporation of existing C libraries
- Simple Domain Specific Language creation
- Support for REST and SOAP web APIs
- Distributed programming, through a simplified Java RMI style library

Why Ruby?
Ruby is already finding a place in the scientific community and industry:

- MPI extension for quick prototyping
- New libraries support MapReduce
- The popular Ruby on Rails framework supports RESTful development

Ruby was originally developed by Yukihiro Matsumoto (Matz), based on his favorite languages. The Ruby language:

- Is purely object-oriented and dynamically-typed
- Provides open classes, allowing meta-programming
- Provides simple iterators through Ruby blocks

The Challenge
The power of Ruby comes at a price. The current C-based interpreter, the Matz Ruby Interpreter (MRI), suffers poor performance since the abstract syntax tree (AST) is used directly by the interpreter. The flexibility of Ruby complicates compilation because context affects the meaning of expressions, as illustrated below.

```ruby
class Fixnum
  def fact
    if self == 0
      1
    else
      self * (self - 1).fact
    end
  end
end

5.fact  =>  120  # expected result
```

Safe methods are those that do not perform I/O and do not rely on global variables. A tool to perform the analysis of the C code was developed, both to speed the analysis process, and insulate against changes in Ruby. Figure 3 illustrates how C source files are analyzed.

The Algorithm
Our algorithm first analyzes, then uses the results to partially evaluate Ruby source, producing a new Ruby file. Currently, the partial evaluator targets whole programs due to Ruby’s dynamic nature, though we are confident it can be extended to handle libraries. Analysis proceeds top-down from the beginning of a program, following a traditional data-flow approach.

Method Call Safety
Safe method calls do not access shared state, have safe blocks, and only call safe methods, though we relax the shared state requirement by taking an “all-or-nothing” approach, marking methods safe, only if all methods accessing a shared variable are safe. Analysis runs iteratively, until a fixed point is reached, ensuring information about shared state is as complete as possible.

Special Cases
Object Initialization The special method new first allocates space, then calls the initialize method for newly created objects. The partial evaluator must ensure initialize calls only safe methods.

Loading Modules Ruby loads supporting modules through the load and require methods, loading files into the top-level environment, unless a module is specified. The partial evaluator must mimic this behavior, analyzing loaded source code as if it were part of the original program.

Future Work
Once the partial evaluator is complete for whole programs, we hope to use the insights gained during development to:

- Partially evaluate modules, inserting runtime checks to ensure correctness
- Analyze C extensions to Ruby, exposing their semantic properties to the partial evaluator
- Perform further specialization, such as method-inlining

References