DNote: Generating Type Annotations for Ruby Programs using DRuby
CMSC631 Spring 2009

Tandeep Sidhu    Mark Daly

May 11, 2009
An Introduction to DRuby

- Static type checker for Ruby [1]

```ruby
def add3(x)
    x + 3;
end
puts add3("info");
```

DRuby: [ERROR] instance Fixnum does not support methods to_str

- **Process**: AST, Ruby Intermediate Language (RIL), Constraint Graph
- Type Annotations
Type Annotations
Also known as Type Signatures

OCaml:

```ocaml
let add3 (x : int) : int = x + 3
```

Ruby/DRuby:

```ruby
def add3(x)
  x + 3
end
```

```
def initialize(*args)
  # specify
  if 4 == args.length &&
      args[0].kind_of?(Vector3fc) &&
      args[1].kind_of?(Numeric) &&
      args[2].kind_of?(Integer) &&
      args[3].kind_of?(Integer)
    position, slice3Dcube, nodItemsPerCell, nodLevelCount = args
    @dimensions = OctreeDimensions.new(position, sizeOfCube, nodItemsPerCell, nodLevelCount)
    @rootCell = nil
  # copy
  elsif 1 == args.length &&
    args[0].kind_of?(OctreeElement)
    other = args[0]
    @dimensions = OctreeDimensions.new(other.dimensions)
    @rootCell = Octree.copy?(other.rootCell)
  else
    raise "invalid arguments"
  end
end
```
Goals

• Construct type annotations for methods with “closed” signatures in constraint graph.
  From this:
  
  ```python
  def add3(x)
      x + 3
  end
  
  puts add3(4)
  ```

  Construct this:
  ```markdown
  ##% add3 : (Fixnum) → Fixnum
  ```

• Explore approaches to finding annotations with partial information
A Different Method of Type Inference

- Key Differences from Hindley-Milner type inference
  - All constraints are subtyping relations
    \((\text{Var} \leq \text{Var})\)
  - Global approach to solving constraints
Type information is encoded in the constraint graph. We need to identify and extract constraints on user methods.

Example:

```ruby
class ClassA
  def add3(x)
    x + 3
  end
end
my_a = ClassA.new
print mya.add3(4)
```

Goal annotation: `##% add3 : (Fixnum) → Fixnum`
Finding Complete Types in the Graph

- Use information stored in the graph to materialize type annotations for user methods
- Deconstruct method types and refine through graph traversal (there are several complications not shown by this example)
Current Status

- Working with small test programs/classes
- Can identify and consolidate constraint graph edges corresponding to user methods
- Can “unfold” inferred method signatures and use signature components to re-search the graph (for type refinement)
- Started testing with benchmark programs from [2], helping to guide development
Remaining Tasks/Future Experiments

- Build “pretty-printer” to produce complete annotations from position/signature value pairs
- Find/follow edges from non-closed parameter types
- Test on more benchmark programs, identify instances where this approach fails
- Consider heuristics when information in graph is incomplete (will probably be left for future work)
  - Try to constrain irresolvable type variables with all known concrete classes, provide signatures for types that unify
  - Try to build “witness” class that satisfies constraints imposed on irresolvable parameter/return types
M. Furr, D. An, J. Foster, and M. Hicks.
Diamondback ruby guide.

M. Furr, J. hoon (David) An, J. S. Foster, and M. Hicks.
Static Type Inference for Ruby.
In OOPS Track, SAC, 2009.
To appear.