Dusting up Rubydust

Performance and Testing Improvements

Tim Destan
Michael Whidby

December 6, 2010
What is Rubydust?

- Constraint-based dynamic type inference for Ruby
  - Infer static types from dynamic runs

- dust = dynamic unraveling of static types
Type Inference Review

- Goal is to determine the types of values in a programming language (without the programmer having to write down type annotations)
  - Saves the programmer from having to write down types all the time
  - Hopefully provides type safety almost as good as explicitly declared type annotations
  - Dynamically typed languages usually do the former but not the latter
Type Inference Review (cont.)

- Each unknown value is given a type variable, and these type variables are constrained by the uses of that value.

- These constraints are then unified to find the most general types.
How does Rubydust work?

- Wrap each run-time value to associate with a type variable
- Wrapper generates constraints on this type variable when the wrapped value is used
Simple Example: Input

class B
  infer_types()
  def bar(y)
    y + 3
  end
end

class BTest # < Test::Unit::TestCase
  include RubyDust::RuntimeSystem::TestCase
  def test_bar()
    b = B.new
    x = b.bar(1)
    assert_equal(4, x)
  end
end
Simple Example: Output

argon:src mawhidby$ ./rubydust ../tests/infer/simple_with_base_types/simple.rb
requiring ../tests/infer/simple_with_base_types/simple.rb...
Patching the annotated code...
Patching the unannotated code...
Running the program...
Inferring the types...
Reconstructed types are:

class B;
    str_typesig("bar :(['+'] : (Numeric) -> Numeric]) -> Numeric")
end
class MiniStat::Data;
    str_typesig("q3 :() -> Numeric")
    str_typesig("mode :() -> Numeric")
    str_typesig("partition :
        (Numeric, [each : () -> Array<Numeric>]) ->
        Hash<.?,.?>
    ")
    str_typesig("mean :
        ([each : () -> .?;size : () -> Numeric;inject : (.?) -> .?]) ->
        Numeric
    ")
    str_typesig("harmonic_mean :() -> .!")
    str_typesig("variance :() -> Numeric")
    str_typesig("outliers :() -> Array<.?>")
    str_typesig("std_dev :() -> Numeric")
    str_typesig("median :
        ([sort! : () -> Array<Numeric>;
            size : () -> Numeric;
            '[]' : (Numeric) -> Numeric]) ->
        Numeric
    ")
    str_typesig("q1 :() -> Numeric")
    str_typesig("data :() -> .!")
    str_typesig("to_s :() -> .!")
    str_typesig("initialize :([collect : () -> Array<.?>]) -> Boolean")
    str_typesig("iqr :() -> Numeric")
    str_typesig("geometric_mean :() -> .!")
What are we doing?

- Feature enhancements
  - Porting Rubydust to Ruby 1.9
  - Testing framework

- Performance improvements
  - Improving run time
Porting Rubydust to 1.9

- Ruby 1.9 has removed some libraries from 1.8
  - Test::Unit replaced by MiniTest
    - Can still install Test::Unit via a gem.
    - Built in module still there but some features removed, so need to use the gem version.
  - GetOptLong removed, replaced by OptionParser
    - Rewrote command line option parser to use new library, which is also present in 1.8 versions
Porting Rubydust to 1.9

- Change in 1.9 to LOADPATH - no longer includes current directory by default
  - 1.9 added require_relative method to allow requiring relative to location of the code file.
  - We changed Rubydust's relative referencing to use require_relative and wrote our own for use in 1.8.
  - Ruby's introspection capabilities enable the programmer to check if a method exists and define a new one if it doesn't
Testing Framework

- **Unit tests**
  - Already implemented (provided)

- **Integration tests**
  - In progress
Performance

- Hebruby - Julian and Hebrew calendar date conversion
  - 274 lines of code
  - Execution time: 0m 33.078s

- Ministat - generates statistical info on numerical data sets
  - 130 lines of code
  - Execution time: 1m 15.165s
Recently, another student added profiling to Rubydust
  ○ Using ruby-prof gem (Ruby profiler written in C)
    ○ Can show method times, what calls a method, and what is called by a method

Useful for identifying possible inefficiencies
## Example Profiler Output

<table>
<thead>
<tr>
<th>%total</th>
<th>%self</th>
<th>total</th>
<th>self</th>
<th>wait</th>
<th>child</th>
<th>calls</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>100.00%</td>
<td>0.00%</td>
<td>358.32</td>
<td>0.00</td>
<td>0.00</td>
<td>358.32</td>
<td>1</td>
<td>&lt;Module::Rubyd&lt;Users/tdestan/code/rubydust/src/runtime/init.rb:182&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>/Users/tdestan/code/rubydust/src/run</td>
</tr>
<tr>
<td>282.31</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>282.31</td>
<td>1/1</td>
<td>Rubydust::Cons</td>
</tr>
<tr>
<td>75.95</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>75.95</td>
<td>1/178383</td>
<td>Class::new</td>
</tr>
<tr>
<td>0.06</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.06</td>
<td>1/1</td>
<td>Rubydust::Cons</td>
</tr>
<tr>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>2/2</td>
<td>&lt;Module::Rubyd<a href="">Class::Time</a>::#</td>
</tr>
<tr>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>2/2</td>
<td><a href="">Class::Time</a>::#</td>
</tr>
<tr>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>3/453433</td>
<td>Hash::[]</td>
</tr>
<tr>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>1/1</td>
<td>Time::-</td>
</tr>
<tr>
<td>37.12</td>
<td>8.40</td>
<td>0.00</td>
<td>0.00</td>
<td>37.07</td>
<td>26970/26976</td>
<td>Set::each</td>
<td></td>
</tr>
<tr>
<td>319.12</td>
<td>0.03</td>
<td>0.00</td>
<td>0.00</td>
<td>319.09</td>
<td>6/26976</td>
<td>RGL::DirectedA</td>
<td></td>
</tr>
<tr>
<td>99.42%</td>
<td>2.35%</td>
<td>356.23</td>
<td>8.43</td>
<td>0.00</td>
<td>356.16</td>
<td>26976</td>
<td>Hash::each_key</td>
</tr>
<tr>
<td>343.75</td>
<td>0.22</td>
<td>0.00</td>
<td>0.00</td>
<td>343.54</td>
<td>3929/18131</td>
<td>RGL::DirectedA</td>
<td></td>
</tr>
</tbody>
</table>