

Patterns: (yet another) pattern matching Library

Florian Gilcher <flo@andersground.net>

March 30, 2008

In Short: About me

Florian Gilcher, Student of computer engineering at the University of Applied Sciences in Mannheim, Germany.

I'm practicing Ruby mostly as a (very time-consuming and fun) hobby for about 4 years now.

This work is based on the article „If it's not nailed down, steal it.”
by Topher Cyll and his library „Multi Dispatch”
(<http://multi.rubyforge.org>).

Why another pattern matching library?

There are multiple libraries on Rubyforge that accomplish Pattern matching. Why not use one of those?

Rubyforge name	reason against
multi	Old, proof of concept
matchable	small & uses Rinda??
pattern-match	nice set of features but unreleased
patternmatching	Complex DSL Style

Example: patternmatching

From the patternmatching library documentation by Ryoichi Ichiyama.

```
# match example
def calc(code)
  make(code) {
    seems as {plus(:a, :b)} do calc(a) + calc(b) end
    seems as {mul(:a, :b)} do calc(a) * calc(b) end
    seems something do code end
  }
end

code = build {plus(mul(100, 100), 200)}
p calc(code) #=> 10200
```

Example: patternmatching

From the patternmatching library documentation by Ryoichi Ichiyama.

```
# match example
def calc(code)
  make(code) {
    seems as {plus(:a, :b)} do calc(a) + calc(b) end
    seems as {mul(:a, :b)} do calc(a) * calc(b) end
    seems something do code end
  }
end

code = build {plus(mul(100, 100), 200)}
p calc(code) #=> 10200
```

This is too complex for daily use!

A proposition

Lets define a matching pattern within a method definition.
An easy java-like example:

```
class A
  extend Patterns
  multi_def(:my_method) do
    match(String, String) do |string1, string2|
      1
    end
    match(String, Fixnum) do |string1, number1|
      2
    end
  end
end

instance = A.new
a.my_method "string", "string" #=> 1
a.my_method "string", 1       #=> 2
a.my_method "string", 1.0     #=> Exception
```

How `#multi_def` works

Method definitions are evaluated in a special environment. This gives us the possibility to handle argument lists within contained context without messing with core objects.

```
def (name, environment = Patterns::DefaultEnvironment,
    &definitions)
  def_env = environment.new(self, name)

  def_env.instance_eval &definitions
end
```

The actual method implementations will later be executed using some flavour of `#instance_exec` to avoid having an awkward binding of „self”.

Guards!!!

The definition environment contains a set of classes called „guards” that are automatically mapped to certain constructs in the argument list:

Parameter Type	Guard match Strategy
Class	#kind_of?
Regexp	#match
Hash	checks all keys and values
Proc	#call
Guard	will be used directly

Examples for use of the definition environment

The definition environment gives us the freedom to tinker with almost anything without minding the object that we are defining the method on.

```
multi_def(:weird_method) do
  match( String , -- , &use(:some_implementation))

  match( [String , Fixnum , --] ) { |array| ... }

  match( responds_to(:to_sym , :to_s) ) { |obj| ... }

  match( lambda {|a| a > 1} ) { |number| ... }
end
```

As long as our argument list is valid Ruby and our definition environment can make sense of it, you have a pattern.

Some syntactic sugar

As all argument lists will get evaluated in this context, we have total freedom to use *#method_missing*-tricks:

```
multi_def(:fac) do
  match(0) { 1 }
  match(n | n > 0) { |n| n * fac(n-1) }
end

multi_def(:sum) do
  match([]) { 0 }
  match(x::xs) { |x,*xs| x + sum(xs) }
end
```

Why do we need another style for calling methods?

Why do we need another style for calling methods?

Let's have a look at *ActionController::Base#render* :

```
if file = options[:file]
  render_for_file(...)

elsif template = options[:template]
  render_for_file(...)

elsif inline = options[:inline]
  add_variables_to_assigns
  render_for_text(...)
```

Why do we need another style for calling methods?

Let's have a look at *ActionController::Base#render* :

```
if file = options[:file]
  render_for_file(...)

elsif template = options[:template]
  render_for_file(...)

elsif inline = options[:inline]
  add_variables_to_assigns
  render_for_text(...)
```

This method has 84 lines and is really hard to extend. You cannot add or override behaviour in the middle of the method.

A new implementation

```
multi_def(:render) do
  match :file => String do |options|
    render_for_file(...)
  end
  match keys(:template) do |options|
    render_for_file(...)
  end
  match :inline => _ do |options|
    add_variables_to_assigns
    render_for_text(...)
  end
end
```

This is easily extensible, just call `#match_def` again within the class or a subclass and define another pattern.

- At the moment, calling a matched method instead of a straight-forward implementation is about 15 times slower. Depending on the interpreter, this can go up to a magnitude of 40 (Ruby 1.9.0-1). It crashes Rubinius (anyone in for a debug session?).

- At the moment, calling a matched method instead of a straight-forward implementation is about 15 times slower. Depending on the interpreter, this can go up to a magnitude of 40 (Ruby 1.9.0-1). It crashes Rubinius (anyone in for a debug session?).
- The (somewhat) good news: to this date, I put no large effort in performance.

- At the moment, calling a matched method instead of a straight-forward implementation is about 15 times slower. Depending on the interpreter, this can go up to a magnitude of 40 (Ruby 1.9.0-1). It crashes Rubinius (anyone in for a debug session?).
- The (somewhat) good news: to this date, I put no large effort in performance.

Some random quote

"...faster than three-legged dogs and turtles, but slower than the Space Shuttle."

- Evan Phoenix

- At the moment, calling a matched method instead of a straight-forward implementation is about 15 times slower. Depending on the interpreter, this can go up to a magnitude of 40 (Ruby 1.9.0-1). It crashes Rubinius (anyone in for a debug session?).
- The (somewhat) good news: to this date, I put no large effort in performance.

Some random quote

"...faster than three-legged dogs and turtles, but slower than the Space Shuttle."

- Evan Phoenix

- Patterns is not yet released, I want to document it first. It will be announced on the ruby-talk mailinglist. (<http://patterns.rubyforge.org>)

Shameless Plug

At the moment, I am searching for an internship abroad in context of my studies.

Spoken languages: French, English. I have no preference towards the country.

Time: 6 months, from August/September on.

Details: **flo@andersground.net**

Thank you for listening, I hope you enjoyed the journey.